send check or money order to: People's Computer Company
P.O. Box 310
Menlo Park, Ca 94025

name ____________________
address __________________
____
____
zip

what kind of computer do you use? ______________________

* subscriptions start with 1st issue of school year

$4 for 5 issues
($5 Canada & overseas)

Bulk Rate
U.S. Postage
PAID
Menlo Park, CA.
Permit No. 371

TO: SERIAL SUPERVISOR
ERIC CLEARINGHOUSE
D 10 CYPRUS HALL
STANFORD UNIVERSITY
STANFORD CA 94305

PCC

is a newspaper about . . .

having fun with computers
learning how to use computers
how to buy a mini-computer
books, films, music
tools of the future.

Help us write the next issue . . .

- *do you have a computer?*
- *do you like your computer? (do you like the computer manufacturer?)*
- *do you have a music composer program?*
- *how do you build a cheap tape winder?*
- *do you have any good game playing program or simulations (in BASIC)?*
- *what would you like to see in this newspaper?*
- *would you like to do one or more pages for a future issue?*

Contributors –
Roy M. Worthington
Kids from LHS

We did this issue –

Bob Albrecht
Jerry Brown
Marc Le Brun
Peter Lynn Sessions
LeRoy Finkel
Jane Wood
Mary Jo Albrecht

Many persons who are not conversant with mathematical studies imagine that because the business of [Babbage's Analytical Engine] is to give its results in numerical notation, the nature of its processes must consequently be arithmetical and numerical, rather than algebraical and analytical. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; and in fact it might bring out its results in algebraical notation, were provisions made accordingly

~ ADA AUGUSTA, Countess of Lovelace (1844)

Fundamental Algorithms. Donald E. Knuth. Addison-Wesley Publishing Co., 1968

Quantity prices for this issue –

1 – 9	$1.00 each
10 – 99	$0.60 each
100 or more	$0.35 each

Please send ______ copies of the December 1972 issue of PCC.

Amount enclosed $ __________

Name ____________________

Address ____________________

____

____

Zip

CONTENTS:

(SEE INSIDE)

PEOPLE'S COMPUTER COMPANY is published 5 times during the school year. Subscriptions begin with the first issue, October 1972.

Single subscriptions – $4 for five issues ($5 Canada and overseas)

Group subscriptions, mailed Book Rate, all to the same address –

10 or more	$3.00 each
30 or more	$2.50 each
100 or more	$2.00 each

Subscription coupon on back cover.

people's computer center

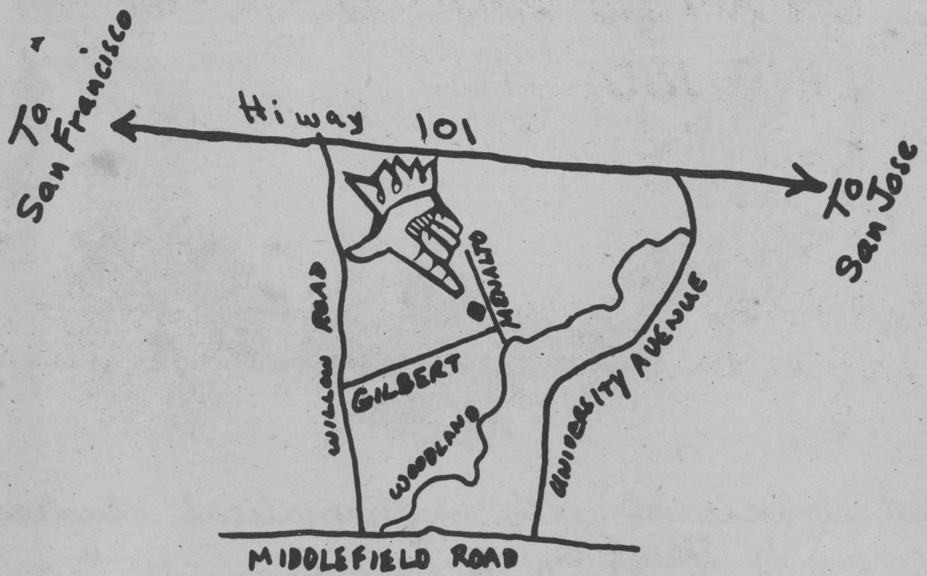
is a place

. . . a place to do the things the People's Computer Company talks about.

. . . a place to play with computers at modest prices.

. . . a place to learn how to use computers.

We have a small, friendly computer – a DEC EduSystem 20 with two terminals – a TTY with an acoustic coupler (see page 12) so we can talk to an HP 2000C – some electronic calculators – a programmable toy car – games – book to help you learn and ... we run after school workshops for kids 9 years and up, and evening open classrooms for anyone who wants to participate. We also sell raw computer time for $2 per hour for an EduSystem 20 terminal running in BASIC. Here we are



What's a Dymax?

Yes, the name did come from Bucky Fuller's Dymaxion. Why? Who knows?

We've been in business for 5 years. We number between 4 and 10 people depending on who you count, on which day, plus a group of available relatives. We vary in scope from a straight high school teacher, to a mad scientist-type to a chief fantasist to a few assorted computer freaks.

We do all sorts of things. Primarily, we write good instructional materials (workbooks or the like) for computer and calculator manufacturers, publishers or for our own publication. We also teach lots of people how to use computers through our University Extension courses or in-service activities. We also do hardware consulting and participate at related conferences.

That's us (right now) in a nutshell. What are you?

Next Issue -

In the next few issues we will spend some time giving tips on how to write bid specifications for mini-computer systems for classroom use. We've had some interesting experiences to pass on to you. If you have recently gone through this, please send us copies of your specs, the bids received, and your ultimate choice. Add any comments, warnings, etc. that we can pass on to readers.

We're interested in both software and hardware specs. We'll name names where appropriate and try to give you the true scoop on this very delicate process known as "going to bid."

Huntington Workshop

For San Francisco Bay Area readers, there will be a Huntington Project workshop held at Lawrence Hall of Science, SATURDAY, FEBRUARY 24, 1973, from 9 to 12 AM. The workshop is designed to introduce you to all the Huntington Project simulations and give experienced users a chance to share their ideas.

For a $3.00 per person lab fee, you will receive some project materials and be guaranteed computer time to run the programs.

Remember – the Huntington materials work well in science, social studies and business classes . . . bring along a friend from another department.

Dick Nodlinski, San Ramon High School
LeRoy Finkel, Ravenswood High School
Dave Pessel, UC (originally with the project)

BABSON COLLEGE WOULD LIKE TO EXCHANGE COMPUTER PROGRAMS WITH OTHER SCHOOLS. THEIR PUBLIC LIBRARY HAS SOME 200 PROGRAMS AVAILABLE ON AN EIGHT PAGE LISTING. MANY OF THEIR PROGRAMS ARE STANDARD HP STUFF BUT THEIR LIST CONTAINS SOME OTHER INTERESTING POSSIBILTIES.

IF YOU'RE INTERESTED CONTACT:

EDGAR CANTRY
ACADEMIC COMPUTER SERVICES
BABSON COLLEGE
BABSON PARK, MA. 02157

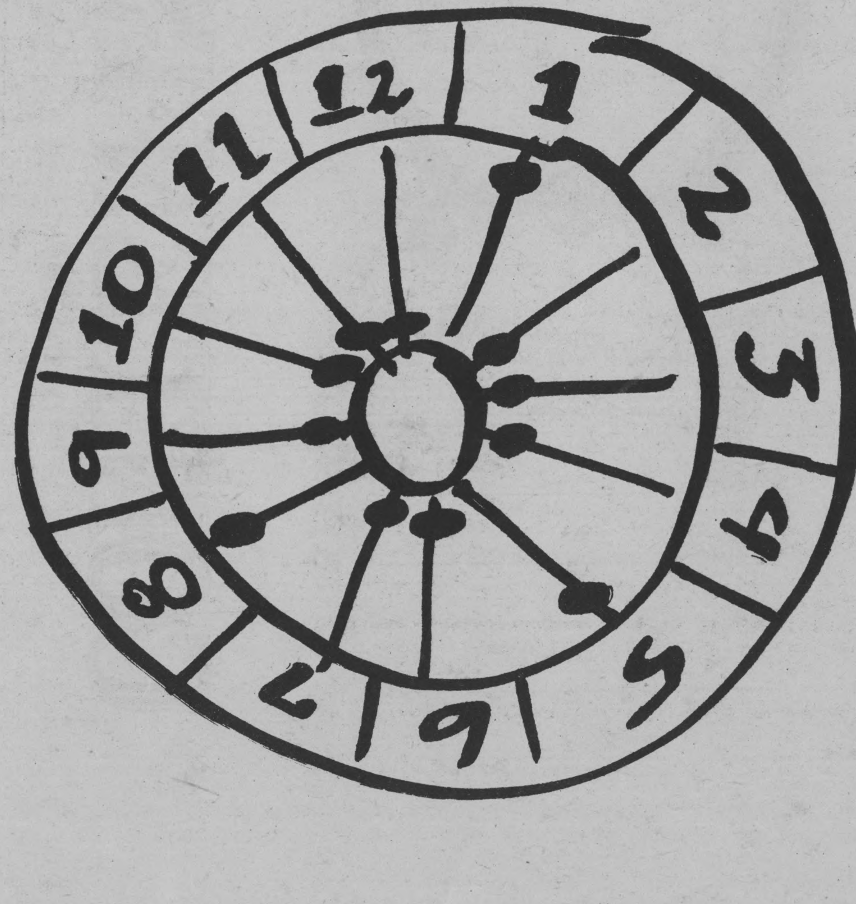
the electric bead game

A series of evenings devoted to music games that you can learn to write yourself and play on our computer.

The sessions are problem/project oriented, with hands-on use of the computer beginning with the first session. Most of the time we will be concentrating on music, but you are encouraged to pursue your ideas in any area that interests you. By the end of the series you will have tackled several projects, perhaps culminating in something BIG, like a program that simulates a composer at work.

Who's responsible for this? Peter Lynn Sessions and Marc LeBrun. Marc and PLS joined forces to develop the BEAD GAME at Portola Institute in menlo park. With Portola's support (kudos to Dick Raymond), the infamous pair achieved a Pyrrhic victory over traditional methods for presenting the materials of music. The Electric Bead Game combines this easy approach to music with PCC's experience in turning people on with computers.

(See also, pages 14 thru 17)



STARS ba

STARS is a number guessing game. We wrote STARS for small people, but big people like it too.

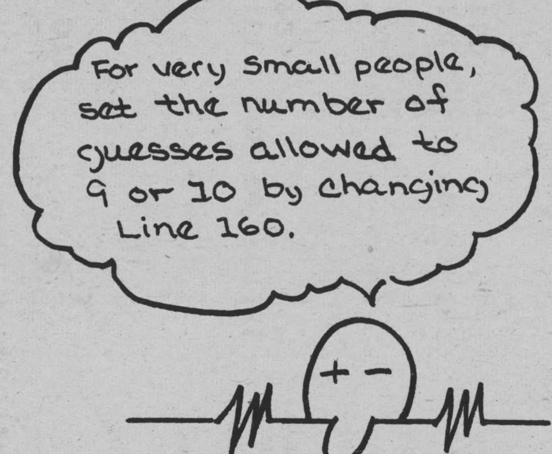
```
100  REM *** STARS - PEØPLE'S CØMPUTER CENTER, MENLØ PARK, CA
110  PRINT "STARS - A NUMBER GUESSING GAME"
120  PRINT
130  RANDØM
140  REM *** A IS LIMIT ØN NUMBER, M IS NUMBER ØF GUESSES
150  LET A=100
160  LET M=7
170  PRINT "DØ YØU WANT INSTRUCTIØNS (1=YES 0=NØ)";
180  INPUT Z
190  IF Z=0 THEN 280
200  REM *** INSTRUCTIØNS ØN HØW TØ PLAY
210  PRINT "I AM THINKING ØF A WHØLE NUMBER FRØM 1 TØ";A
220  PRINT "TRY TØ GUESS MY NUMBER. AFTER YØU GUESS, I"
230  PRINT "WILL TYPE ØNE ØR MØRE STARS (*). THE MØRE"
240  PRINT "STARS I TYPE, THE CLØSER YØU ARE TØ MY NUMBER."
250  PRINT "ØNE STAR (*) MEANS FAR AWAY. SEVEN STARS (*******)"
260  PRINT "MEANS REALLY CLØSE! YØU GET";M;"GUESSES."
270  REM *** CØMPUTER 'THINKS' ØF A NUMBER
280  PRINT
290  PRINT
300  LET X=INT(A*RND(0))+1
310  PRINT "ØK, I AM THINKING ØF A NUMBER. START GUESSING."
320  REM *** GUESSING BEGINS. HUMAN GETS M GUESSES.
330  FØR K=1 TØ M
340  PRINT
350  PRINT "YØUR GUESS";
360  INPUT G
370  IF G=X THEN 600
380  LET D=ABS(X-G)
390  IF D >= 64 THEN 510
400  IF D >= 32 THEN 500
410  IF D >= 16 THEN 490
420  IF D >= 8 THEN 480
430  IF D >= 4 THEN 470
440  IF D >= 2 THEN 460
450  PRINT "*";
460  PRINT "*";
470  PRINT "*";
480  PRINT "*";
490  PRINT "*";
500  PRINT "*";
510  PRINT "*";
520  PRINT
530  NEXT K
540  REM *** DID NØT GUESS NUMBER IN M GUESSES
550  PRINT
560  PRINT "SØRRY, THAT'S";M;"GUESSES. NUMBER WAS";X
570  PRINT "LET'S PLAY AGAIN. "
580  GØTØ 280
590  REM *** WE HAVE A WINNER
600  FØR N=1 TØ 50
610  PRINT "*";
620  NEXT N
630  PRINT "!!!"
640  PRINT "YØU GØT IT IN";K;"GUESSES!!! LET'S PLAY AGAIN."
650  GØTØ 280
660  END

RUN


STARS - A NUMBER GUESSING GAME

DØ YØU WANT INSTRUCTIØNS (1=YES 0=NØ)?1
I AM THINKING ØF A WHØLE NUMBER FRØM 1 TØ 100
TRY TØ GUESS MY NUMBER. AFTER YØU GUESS, I
WILL TYPE ØNE ØR MØRE STARS (*). THE MØRE
STARS I TYPE, THE CLØSER YØU ARE TØ MY NUMBER.
ØNE STAR (*) MEANS FAR AWAY. SEVEN STARS (*******)
MEANS REALLY CLØSE! YØU GET 7    GUESSES.


ØK, I AM THINKING ØF A NUMBER. START GUESSING.

YØUR GUESS?50
**

YØUR GUESS?75
*

YØUR GUESS?25
***

YØUR GUESS?35
***

YØUR GUESS?40
***

YØUR GUESS?20
****

YØUR GUESS?10
*******

SØRRY, THAT'S 7    GUESSES. NUMBER WAS 9
LET'S PLAY AGAIN.


ØK, I AM THINKING ØF A NUMBER. START GUESSING.

YØUR GUESS?32
****

YØUR GUESS?40
***

YØUR GUESS?24
*****

YØUR GUESS?20
******

YØUR GUESS?18
**************************************************!!!
YØU GØT IT IN 5    GUESSES!!! LET'S PLAY AGAIN.


ØK, I AM THINKING ØF A NUMBER. START GUESSING.

YØUR GUESS?
```

STARS WAS WRITTEN FOR EDUSYSTEM 20. TO RUN IT ON AN HP 2000, DELETE LINE 130.

EACH PRINT STATEMENT (LINES 450 TO 510) PRINTS ONE STAR AND ALSO RINGS THE BELL ON THE TTY.

- Play it for fun.
- Play it many times – figure out what the stars mean.
- Develop a strategy so that you can always guess the number in at most 7 guesses.

??? Is there a strategy that will guarantee always guessing the number in at most 6 guesses?



NEXT ISSUE –

we will publish listings of several number guessing games.

In the last issue we introduced you to the Huntington Project materials but failed to list a source for these materials (failed only because the information was slow in coming). Now we have the details and the news is GOOD.

Digital Equipment Corporation (DEC) is now the sole source of the latest Huntington materials. DEC's price is ridiculously low and any profits they make will be given to the project to cover future development costs. As you can see from the price sheet, you can buy whatever materials you need in any combination you want. Take advantage of the discounts, they're worth it ... and be sure to buy the paper tape for each program.

If you care to act as a critic, make suggestions, donate ideas, or whatever, you should write the project director —

Dr. L. Braun, College of Engineering, State University of New York at Stony Brook, Stony Brook, NY 11790

DEC would like us to point out that the original Huntington Project programs are available on paper tape for $1 per tape. $50 buys you the whole set (and saves endless hours on a terminal). These are written in very standard BASIC and should run on most any system with (at worst) minor editing.

✓ Available now. Others available soon — contact DEC for exact dates.

HUNTINGTON TWO SIMULATION PACKAGES

Biology	
✓ GENE1	Simulates the inheritance of genetic traits.
HARDY	Simulates the Hardy-Weinberg Principle population genetics.
✓ LOCKEY	The lock and key model of enzyme action.
MALAR	Control of malaria epidemic.
✓ POLUT	Simulates pollution of a lake, river, or pond from various types of waste.
✓ STERL	Insect control by means of sterile males vs pesticides.
TAG	Sampling exercise on a simulated farm pond.
Physics	
✓ CHARGE	Determine electron charge with simulation of Millikan's oil-drop experiment.
FRANCK	Simulation of Franck-Hertz experiment.
GRAV	Simulation of gravity based on Cavendish experiment.
QUANTA	Simulation of Einstein/Planck experiments showing light to consist of energy quanta or photons.
REACTR	Simulates operation of a nuclear reactor.
SCATER	Alpha particle scattering to determine atomic structure.
✓ SLITS	Young's double slit experiment showing light interference patterns.
Social Studies:	
BALPAY	Simulates U.S. balance of trade payments.
BUFLO	Herd population management of buffalo or other threatened species.
ECON/GNP	Simplified model of U.S. economy.
✓ MARKET	Game engaging two companies in one-product competition.
MASPAR	Model of mass participation in decision making.
✓ POLICY	As members of various pressure groups, students formulate national policy.
✓ POLSYS	As various community members, students influence city government policy.
POP1-5	Series of simplified population models.
Miscellaneous:	
SAP	Statistical analysis package.

PRICING

HUNTINGTON TWO SIMULATION MODULES

Individual Components

Student Lab Manual	$0.30
Teacher Commentary and Instructions	.30
Background Resource Materials	.50
Paper Tape of Program	1.00
Individual Packet	2.10
Individual Packet (no paper tape)	1.10

Classroom Packets (10% discount over individual component price)

Student Lab Manual (40), Teacher Commentary (2), Resource Material (4), Paper Tape (1) Classroom Packet	$14.00

Quantity Orders (10 or more—20% discount over prices above)

10 Individual Packets ($2.10 x 10)	$21.00
— 20% discount	4.20
Total, 10 Individual Packets	16.80
Total, 10 Packets (no paper tapes)	8.80
10 Classroom Packets ($14.00 x 10)	$140.00
— 20% discount	28.00
Total, 10 Classroom Packets	112.00

HUNTINGTON ONE PROJECT BOOKLETS

Biology (7 simulation programs)	$1.00
Earth Science (4 simulation programs)	1.00
Chemistry (12 simulation programs)	2.00
Physics (21 simulation programs)	2.00
Social Studies (5 simulation programs)	1.00
Mathematics (20 application programs)	2.00
Teacher Assistance (8 application programs)	1.00

Quantity Orders (20 or more of the *same* booklet—20% discount on prices above)

market

One of our favorite Huntington Two programs is MARKET, "a computer based game of competition between two companies selling the same type of product." Participants control the production level, advertising budget and unit price while the computer model grinds out the results of the competition. There are a number of random "events" that occur to spice up the game. You can play as long as you want or until one company (team) reaches assets of $12 million or until one company goes bankrupt.

You can use MARKET as a game without any previous study (you'd be amazed at how much learning goes on). Or you can use MARKET as a complete learning unit and simulation activity. The resource handbook teaches all about business operations, supply and demand, advertising efforts and other business-type information. It contains a very complete explanation of the entire MARKET simulated economy. It would be great in a business or social studies class.

We suggest this as a "team" activity. The exchange of ideas about strategies between the participants on each team is as important as the win or loss.

Here's a sample RUN for your reading pleasure.

```
QUARTER  8

PROFIT  MARKET SHARE  CASH ON HAND  NUMBER SOLD  INVENTORY  ASSETS
 521       45.94         10220          34           46       11232
 559       54.05         10012          40           55       11222

THERE HAS BEEN A STRIKE AND YOUR PRODUCTION HAS BEEN
HALTED. NEGOTIATIONS HAVE BEEN STARTED, BUT HOPE OF
A SETTLEMENT LOOKS DIM.


COMPANY  1
NO PRODUCTION POSSIBLE DUE TO STRIKE
ADVERTISING BUDGET?25
UNIT PRICE?55

COMPANY  2
NO PRODUCTION POSSIBLE DUE TO STRIKE
ADVERTISING BUDGET?50
UNIT PRICE?55


QUARTER  9

PROFIT  MARKET SHARE  CASH ON HAND  NUMBER SOLD  INVENTORY  ASSETS
-490       48.88         10166          22           24       10742
-596       51.11         9858           23           32       10626

STRIKE SETTLED. NORMAL PRODUCTION RESUMED
NEW VARIABLE COST=$ 24    / UNIT DUE TO INCREASED WAGES
```

****

Fixed production cost = $250000./quarter
Variable production cost = $20/unit
With no advertising and a selling price of $50/unit a company will sell 250000 units (printed as 25)
Warehouse charge for inventory = 5 per cent
Interest charge on borrowed money = 5 per cent

Units and dollars are in thousands

****

HUNTINGTON

Materials may be ordered from —
Mr. Tom Mullaine
Software Distribution Center 1–2
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

A POLEMIC by marc le brun:

TILTING AT WINDMILLS, OR WHAT'S WRONG WITH BASIC?

BASIC *vs. all those other languages*

For you FORTRAN, APL and COBOL fanatics who are upset at our s–t–r–o–n–g stand for BASIC, we owe you an explanation. We deal primarily with school age kids and prefer BASIC for any or all of these reasons:

BASIC is easily learned by large numbers of students. (We want to see many kids involved,not an elite few.)

BASIC is easily learned by people with a wide range of ability (you don't need a strong math background to learn it).

You can learn BASIC quickly. You don't need hours of instruction before you can write your own simple programs.

There are a large number of instructional materials available written for novice computer users (teach yourself style, not programmed instruction).

There is a tremendous wealth of support materials written to support BASIC in a variety of classroom situations (Huntington project, Project SOLO, Denver Calculus Project, REACT, HP and DEC materials).

The extensions to BASIC now available on nearly any disk, time sharing system make it possible to do nearly anything in BASIC that you might want to do in another language, though BASIC may not be as efficient.

BASIC is available in a time sharing environment on many very low cost computer systems.

Society to
Help
Abolish
Fortran
Teaching
SHAFT

P. O. Box 310
Menlo Park, Calif. 94025

Now, before you hasten to write and apply these same principles to the language of your choice, keep in mind our target – KIDS – kids ages 8 to 18 in a problem-solving, simulation and gaming environment. We're not concerned with huge number crunching applications nor do we care about super-complicated data processing schemes for which there are many languages far superior to BASIC. We just hope to introduce a large number of new people to a super, fun, learning/recreational tool and want to do it at a reasonable cost and without a great deal of fuss.

For the record, we have recently started doing some work in PILON (PYLON), a subset of PILOT, in our environment. A future issue will give details of our successes or failures. We are also very curious about LOGO and its possible use in our environment. Our friends at Lawrence Hall of Science are working on that and we hope to report on their progress as time goes on. We admit that we are BASIC BIGOTS, but we're looking at other languages as well ... but not FORTRAN, APL, or COBOL!

Leroy F.

It may come as a surprise to many educational/recreational computer users that the language they have come to know and love has serious flaws. By a flaw I mean something fundamentally wrong, as opposed to what could be called defects – problems which could be solved or somehow avoided.

Every actual computer language has its very own garden of defects, which everybody talks about but nobody does anything about. By actual computer language I refer to ones which are actually implemented on some computer somewhere, as opposed to theoretical languages, or specifications for proposed languages, which occur frequently in the literature of computer science. A defect in a language usually results in users complaining loudly about such things as "It won't let me use zero for a subscript in my arrays" or "It won't let me use the bell in PRINT statements" and so on.

Defects are annoying but not serious, they can be corrected by any reasonably competent programmer, assuming there is adequate documentation for the language, which there usually isn't, which is why manufacturers can sell "better and better" versions of the same language (and possibly additional hardware as well) to users who have become hopelessly dependent on the language. Like drug addiction, the first one is usually free.

Flaws in a language are much more dangerous, because the novice user does not notice them. They insidiously distort his views of what computers are capable of, what he is capable of doing with computers, and he is well on the way to joining those tragic figures who sit for hours, backs hunched, eyes glazed, nodding over their terminals: the "experienced" users.

I shall now treat the following topics in turn; What is right about BASIC; What is wrong about BASIC from the standpoint of education; What is wrong about BASIC from the standpoint of recreational use, and finally; What is wrong about BASIC from the standpoint of computer science.

Many of you may have noticed that this periodical has adopted BASIC as a sort of *lingua franca.* This is mainly for pragmatic reasons, and the purpose of this polemic is, in part, an attempt to temper what otherwise might be taken as an implicit whole-hearted acceptance of BASIC. As far as I can tell, the following things are more or less right about BASIC.

It's popular.

An increasing number of people are using BASIC, and, despite problems with incompatible versions, they are able to communicate, and exchange ideas and programs with each other, even though they have extremely different orientations.

It's interactive, more or less.

BASIC was designed to be used in a time sharing environment, so the mechanics of writing and running programs are fairly straightforward, within the limitations of the system it's implemented on, anyway. The I/O statements (INPUT and PRINT), though limited, make the writing of interactive programs relatively easy, which makes it one of the few languages suitable for recreational purposes.

It's widely available.

Many manufacturers offer it, due to an ever growing demand for it by the educational segment of the market (as well as a not insignificant portion of the industrial). Besides, if a manufacturer provides software he also influences how much hardware the consumer buys: "But with the addition of another 32K of core memory and 16 discs you can run our WALLAPALOOZA BASIC ... " The first one is free. He can also sell you software that no self-respecting programmer would admit having written, and the consumer never suspects that he is using software which requires twice as much hardware, services half as many users, has more bugs and fewer capabilities than it could have. Anyway, BASIC is widely available, *caveat emptor.*

It doesn't rock the boat.

BASIC fits in real smooth with the views most people have about what computation is. It introduces them to computers without forcing them to undertake radical restructuring of their concepts. This helps make computers more acceptable, and in a very real sense more accessible to a large number of people, which is good. One of the important things about technology however, is that it forces people to adopt mental models of the world and how it works which are better aligned with reality. More about this in the next section.

Because of the increasing utilization of BASIC in the educational field the following criticisms are particularly relevant. From an educational standpoint the things wrong with BASIC are.

It doesn't rock the boat.

As mentioned above, BASIC is not a mind-expander. It certainly expands the class of experiences and possibilities of anyone who uses it, but this is mainly due to exposure to the computer, and not to BASIC itself. Any system or structure which embodies fundamental principles in a coherent manner almost always causes significant transformations in the habits of thought of those exposed to it. For example, the language LISP, which is directly derived from the fundamental ideas of predicate calculus and recursive functions (more on this later) has a peculiar phenomenon associated with it which I call the "LISP trance." After overcoming the initial difficulties assimilating the principles behind the language the student or new user often finds himself perceiving the world in terms of these basic ideas. A tree is a recursive function applied to a seed, it's branches are a list structure, leaves become "atoms" (specialized meaning) and so on. This state can last as long as a week sometimes. The "cause" of this phenomenon is that the student *has learned patterns which are fundamental* enough to cause noticable restructuring of his internal environment. The new user "comes out of the trance" when this restructuring process has leveled off to the point where the changes involved are imperceptable in relation to the adaptations necessitated by daily life. Since BASIC contains few patterns of any significance (and even those in a muddled sort of way) it is easily used as an instructional medium. It is not necessary for teachers to retool their antiquated and inadequate concepts in order to use it to transfer their antiquated and inadequate concepts to their students. This is not to the student's ultimate benefit.

It creates false impressions.

Besides the theoretically unsound patterns BASIC embodies, which will be discussed later, the language has two other flaws which generally cause the user to have limiting misconceptions; First, there is nothing in it which gives the user any information or facility with computer *systems* in general. Especially deficient in most implementations are the means of dealing with peripheral devices. **(Did you know that in many third-generation computers *every* device is a peripheral, including the CPU and the memory, and the "computer" is really the device which connects and coordinates all the other devices?)** Secondly, "strings" to the contrary (very few languages have real string manipulation features), it forces user to think numerically. I have nothing against numbers, or numerical analysis as such, but numerically oriented processes are but a small portion of the

EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT BASIC BUT WERE AFRAID THAT YOU COULDN'T UNDERSTAND...

entire space of computational processes, and there are many significant, useful and fun things that can be and are done without using either numbers or strings. **(Did you know that inside the computer the BASIC you use probably spends as much as 80 or 90 percent of its time performing non-numeric computations?)** For example, the fields of computational linguistics, operating systems design, and artificial intelligence are concerned almost exclusively with non-numeric processes. What, you ask, could a language be "about" besides letters or numbers? We will discuss this in the theory section. BASIC provides the interested student with very little background if he is to actually use computers in any significant way in the future, as well as causing him to adopt faulty thinking habits.

It's inhibiting.

BASIC restricts exploration and experimentation, and thwarts attempts at self-directed learning, unless the learner is very highly motivated. It is not natural to express or describe processes which are not primarily numeric in nature in the BASIC language, and this implicitly restricts the kinds of uses to which it can be put. Numbers are not all that interesting to most people, and a steady diet of BASIC will never indicate to anyone the most powerful and general view of what a computer is; a *symbol processor*, that is, a tool for manipulating conceptual objects as opposed to tools which manipulate physical material. Consequently the faithful BASIC user is unlikely to discover on his own the full potentialities of the phenomenon of computation.

Many of the observations in the previous section apply with little modification to those users who are primarily concerned with the recreational use of computers. "Recreational" is a somewhat vague adjective, usually meaning "interested in games" or "interested in art." Fairly interesting games can be produced in BASIC, unless the computer is to compete on an equal basis with human players, in which case I advise you to learn some computer science, explore the literature on artificial intelligence, and forget about BASIC entirely (assuming, of course, that the games you're interested in are non-trivial). Also, many of the mathematical features in BASIC are so much dead weight as far as game playing is concerned. For the artist the situation varies depending on the media he wants to work in. There are two ways in which computers can be used in the creative process; design and experimentation, and execution of the work itself. For most users the only type of art that is even remotely possible to directly create with the machine are line drawings, and that only if you have a plotter. Design and experimentation is the most accessible area for exploration at the present time. The inadequacy of BASIC as a medium for expression of non-numeric concepts is clearly a severe limitation here. Again, if you are interested in having the machine "make up" works of art of any significance, abandon BASIC, do your computer science homework and read up on artificial intelligence. I have done a number of things in the area of computer music, one of which was a program that composed original four voice compositions. Your author now divulges a dark secret of his dismal past – *the first computer language I ever used was BASIC!* In the context of this article this is indeed a painful revelation, but it should lend some weight to my argument, since I have sat on both sides of the fence, so to speak. Anyway, the aforementioned program was written in BASIC, and one of the best available implementations at that, (I will not reveal *which* implementation firstly because I don't wish to plug *any* BASIC systems, and secondly because it was done on stolen time, only the names were changed to protect the guilty). Including debugging, it took several months to complete; had it been written in a more suitable language (such as ALGOL) it would have taken at most two weeks, even accounting for my substantial inexperience in things computational at the time. To summarize then; BASIC, while workable to a limited extent, is nearly useless to the incipient computer artist. Then again, so are most other currently available languages.

Before leaving the subject of computer art I would like to make a few brief technical comments on making "teletype pictures:" First, the best ways of doing it are fairly complicated; Second, find out if your BASIC system will let you PRINT a thing called a "bare carriage return," which lets you type characters on top of each other (sometimes you can do it with TAB, assuming of course you *have* TAB on your system, . . .); Third, even the best teletype pictures look terrible, and if you keep doing it hair will grow on your palms and; Fourth, if you persist on this road to esthetic nitwitdom, you will be aided and abetted by the author in an article on Teletype Plotting in a future issue of this amoral periodical.

It is now my intention to examine the "rotten apple which spoiled the bunch," the things that are wrong about BASIC from the standpoint of computer science. It is in this area that the worst flaws exist; hopefully the following comments will also give some indication of what better languages are like.

BASIC has a regular rat's nest of flaws in the area called *control structures* in the language of computer science. What's worse, these problems interact with each other, so it is somewhat difficult subject to untangle. Nevertheless, in the interests of clean living, we shall attempt to thread our way through them.

Imagine your favorite program is RUNning merrily along. At any moment the computer is executing some particular line. Call that line the *active line* and pretend all the other lines are just hanging around waiting to become active when the computer gets to them. The computer does whatever you told it to do in that line. Now if we look at a smaller segment of time we find that the computer is not doing everything on that line all at once, instead it's executing some part of the active line, such as adding two numbers together. We could look at even smaller segments of time, but we would wind up looking at machine language or logic circuits or electronics or quantum mechanics, depending on how small we got, and none of those things concern us here.

We can also look at larger segments of time, and we see that whole big chunks of our program become active together, that is, when one of the lines in the chunk becomes active then you know that all the other lines in the chunk are going to become active pretty soon. This happens to the lines inside a FOR-NEXT loop for instance. These "chunks" have a special name in computer science, they are called *blocks*. Now we can sit and just watch the blocks in our program become active, *without paying any attention to what the lines in the block are telling the computer to do.* If we watch the blocks become active for a while, we will see a couple of different things happen: a block will be active and then suddenly another block somewhere else will be active (the computer did a GO TO), a block will become active over and over (that's a FOR-NEXT loop), sometimes one block will become active after a particular block and sometimes a different one will (IF-THEN) etc. **(What do you imagine a GOSUB looks like?)**

If we watch the program for a long time, maybe for several RUNs, we will observe certain regularities in which blocks become active before or after other blocks. These regularities form a pattern or structure, and that is what the control structure of the program is. (Flowcharts are a primitive way to draw pictures of control structures.)

Now if we look at a LISTing of our program we can see that the places or lines in the program where the computer goes to another block are always one of a small set of statements; GO TO, IF-THEN and so on. Statements of this sort are sometimes called *control statements.*

At this point I can show you one of the flaws in BASIC. Why it's a flaw I will explain a little later. Take the program LISTing and look at it. It is possible to figure out what the control structure is, for instance we could take the LISTing and cut it up into blocks and paste them on a big piece of paper and draw arrows between them, to show what can come after what.

Now take another LISTing and cut it into blocks like before, *and then cut off all the line numbers* and mix up all the blocks and then try to make another chart of the control structure like before, Can't do it, right?

Now do exactly what you did in the last paragraph *except don't cut off the very first line number in each block.* Now you can make the chart. Obviously there is something special about those line numbers which makes them different from all the other line numbers in the program. (So what are all those other line numbers for anyway? We'll discuss that pretty soon.)

Now for one last experiment. Cut all the line numbers off a LISTing and save them, and throw the rest of the program away. Now assume you have amnesia, and don't remember all this stuff we just did. Put a mark of some kind next to all the line numbers in the program which are special. Can't tell them apart, right?

Now clean up all those scraps of paper and I will tell you what this cut and paste exercise was all about.

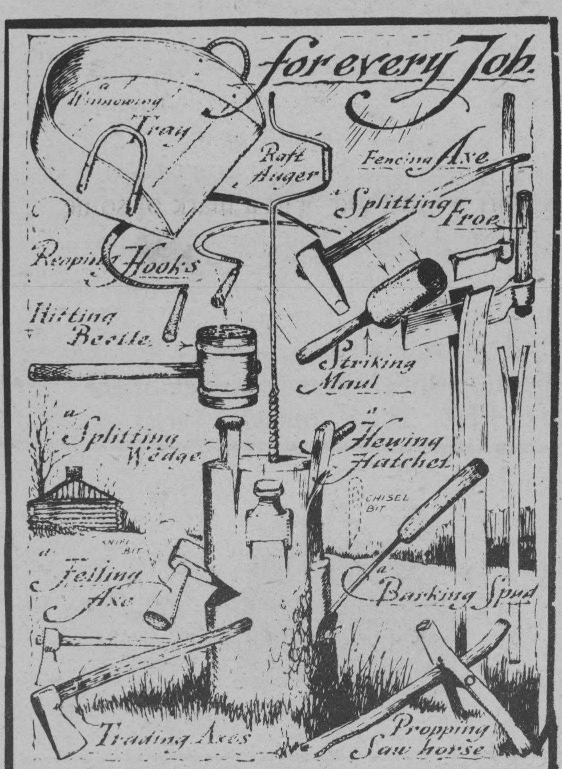
The first computer science flaw we have found in BASIC is this:

The written form of the program gives very little indication of its control structure. Since the control structure reflects how the program operates it is next to impossible to figure out what goes on inside the computer by looking at the program. Since the purpose of a program is to describe what goes on inside the computer, this is clearly a flaw.

In fact, without those special line numbers, there wouldn't be *any* indication at all of the control structure. The only way we could find out is by looking at the "activations" inside the machine, and that is usually just not practical. Those special line numbers have a name in computer science – they are called *labels.* In languages that have labels they are usually written as words, like HERE, HITHER or FOO. Poor old FORTRAN has both line numbers and also labels which are numbers. Never mind, no use beating a dead horse.

Why does BASIC have all those useless line numbers? You are told that line numbers exist so the computer knows what order to do things in, and also so you can insert lines and move them around and stuff. As a matter of fact, line numbers are one of many different means for accomplishing these ends. It is far from the best in terms of user convenience, *but* it is among the easiest to implement when you are designing a language. I do not feel, however, that the needs of the user should at any time be ignored just to make implementation easier. After all, to the user, "a language is forever." Since lines with labels are the only ones the program cares about, all you have to do is remember where they are when you type them in. Then you call all the lines that come between it and the next labeled line things like FOO.1, FOO.2 and so on. If you want to switch them around or edit them later, the EDIT command should be able to tell which is which. But the only time you need those names is when you are typing it in or editing it, they don't have anything to do with the program itself. If you're wondering how you keep all the blocks in the right order, don't worry – *you don't have to.* The only thing that connects them are control statements, and they're already in the blocks. You can think of them as being separated, just like in the chart you made.

Languages that use blocks and labels instead of line numbers are called *block-structured languages.* A very good example of a block-structured language is ALGOL. The idea behind block-structured languages is that the form of the program "on paper" should in some way reflect it's control structure. There is, in addition, another useful thing that can



be done when you have blocks. The blocks can have different "flavors" that is, the language can have different types of blocks to indicate different kinds of processes. BASIC has only one sort of block (in the strict sense) and that is the block defined by FOR and NEXT statements. In languages like ALGOL however, there are many different "flavors" which do what BASIC does with the somewhat messy IF-THEN-ELSE and ON-GO TO instructions, as well as a whole lot of other neat things. So blocks are a means by which the program itself can contain useful information about what it is supposed to do.

You may have noticed that the FOR-NEXT blocks in the program you were examining did not need labels. In fact, you do not, in general, need labels for blocks if their boundries and "flavor" are well defined. You may be wondering at this point if your humble author does not have a few screws loose somewhere, first he goes to great pains to explain all about labels, and then he turns around and says that they're hardly ever necessary. If you've guessed it's because I am about to reveal another flaw, you're correct. So let's get down to business.

Let's make a list of the various control statements that still can't be done by some flavor of block, or else need labels for some reason:

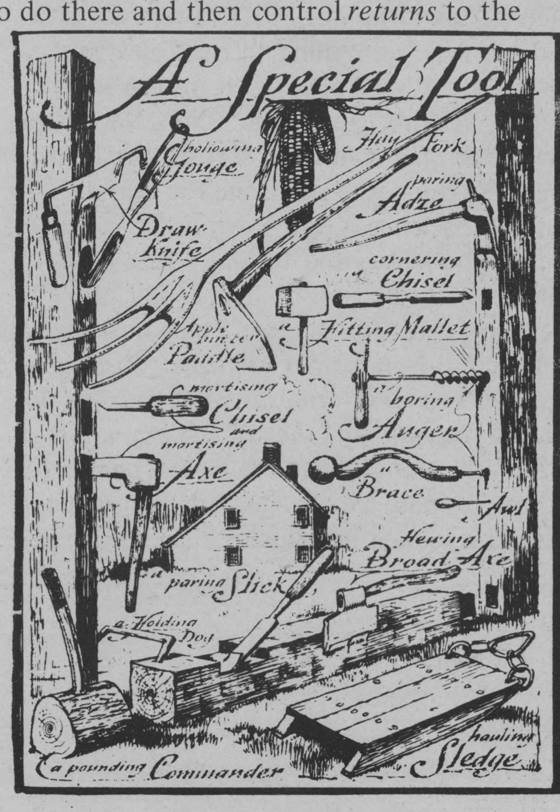
GOSUB (and his sidekick RETURN)
DEF and DIM and
GO TO

For GOSUB the situation seems fairly obvious – we make up a new flavor of block, called say a SUBROUTINE block or a PROCEDURE and we give it a name (label) and we end the block with a RETURN statement. Then if somewhere else in the program we want to call the block we write GOSUB MUMBLE or whatever the name of that block happens to be. OK, that's pretty straightforward, but let me ask you this: Have your programs ever had problems because BASIC sort of "fell" into a subroutine from the statements preceding that section of your program? Not nice is it? Here's computer science flaw number two:

BASIC does not adequately differentiate those sections of the program which are self-contained and meanfully separate and distinct. As a consequence, the "flow" of control into and out of subroutines may deviate radically from its intended and logical "course."

Subroutines are *fundamentally different* from other parts of the program and the language should provide a definite mechanism for delineating them. In addition, control should *absolutely never* pass into a subroutine unless that subroutine has been *explicitly called* from some other part of the program.

You may be surprised at the inclusion of DEF and DIM in my list. You probably don't think of them as control statements at all, but as some other sort of beast entirely. Let's go back to where we made up the SUBROUTINE flavored block. You'll remember that at that point we gave it a name (label). Think for a minute about what subroutines are for; they are special "chunks" of the program that we want the computer to "activate" at various and possibly widely separated places in the program. They perform some useful function and then pass control back to whatever part of the program called them. We indicate the subroutine that we wish to activate by means of its label (name). Now consider what happens when we put one of the FN_ functions in a statement somewhere. The computer goes down the line and various parts of it become active (like when it finds a + or something) and then it gets to our FNX (or whatever we name it). At that point control passes to the line on which we DEFined FNX, the computer does whatever we told it to do there and then control *returns* to the



line that was active at first. As far as control goes, this is *exactly like a subroutine.* This distinction that BASIC makes is completely unnecessary, and exists only because it was convenient for the people that wrote BASIC, certainly not because there is anything good about it as far as the user is concerned. We have "discovered" that BASIC had that flavor block all along; the DEF block. Unfortunately most implementations of BASIC only allow DEF blocks that are one line long.

We have not by any means exhausted DEF as a source of flaws. There is an important difference between calling a subroutine with a GOSUB and calling the one line subroutine attached to a DEF statement using an FN_ type call. GOSUB is just a sort of glorified GO TO in that control and *only* control gets passed to the subroutine and RETURNed to the main body of the program. With DEF, the situation is different – *information as well as control* gets passed between the various parts of the program. Information is passed to the DEF "block" by putting it inside the parentheses that follow the FN_ which calls it. Information is passed back to the part of the program which did the calling in the form of the *value* which the "function" *returns.* The things in the parentheses after an FN_ are called *arguments*, and are said to be *passed* in the same sense as control is said to be passed. Argument passing is an extremely important and fundamental idea in computer science. A value returning subroutine (usually called *procedure*) is also a concept with a similar degree of significance. All of the built in functions in BASIC (such as SIN, RND, etc) are value returning procedures. . . which brings us to computer science flaw number three:

There is no consistent or even adequate mechanism is BASIC for passing control and arguments or for returning values. As a consequence the user must bear the burden of "bookkeeping" which, in fact, should be shouldered by the machine.

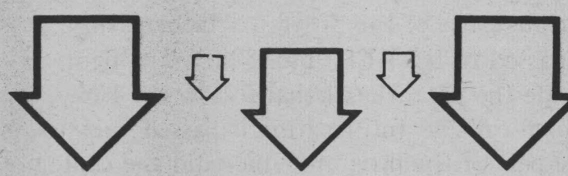
Pardon the "poetry" please. There's more – another fundamental element found in computer science, the *recursive function* is next to impossible to produce in any form whatsoever in BASIC. A recursive function is a function which is defined in terms of itself. Another way of putting it would be to say a recursive procedure is a procedure which calls itself. There is nothing at all "funny" about this idea. If subroutines can call other subroutines, why shouldn't they be able to call themselves as well? Recent results in computer science indicate that the Theory of Computation itself may have very deep roots indeed in the Theory of Recursive Functions. The idea seems to be so important that I will diverge a moment to give a simple example. The factorial function for integers may be defined as follows:

FACTORIAL (X) = 1 if X is zero
X*FACTORIAL(X–1) if X is not zero

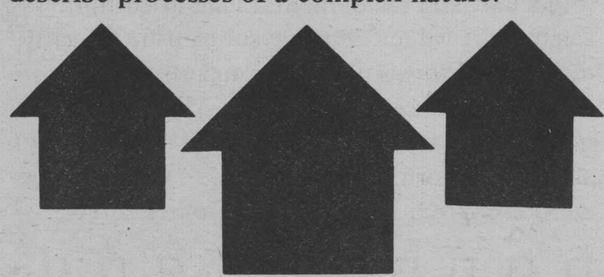
You can see what novice LISP users have to cope with. After you feel you understand this way of defining factorial (as opposed to using a FOR-NEXT loop). Try writing it in BASIC using either DEF or GOSUB. If you can actually get BASIC to swallow it in the first place, try RUNning it to calculate 20! or 30!. If it doesn't blow up consider yourself a natural born programmer, assuming of course, you get the right answer within the limits of accuracy of your system. This is an extremely trivial example for which there exist other and possibly better definitions or calculation methods.

(continued page 8)

There exist however many important useful and fun things which cannot be done any other way. For instance the BASIC you use "figures out" what your lines "mean" and untangles complicated arithmetic expressions by means of a method known as *recursive descent.* I certainly hope so anyway, if it doesn't it's either ten years behind or ten years ahead of other commercially available implementations. There are other, more complicated forms control structures can take, which have mysterious names like *co-routines, parallel control streams* and *mutually recursive procedures,* but I won't go into all that since it can all be summed up as computer science flaw number four:



BASIC has no provisions at all for complex control structures, most importantly there is no means of recursion. The result being that it is a very poor language in which to describe processes of a complex nature.

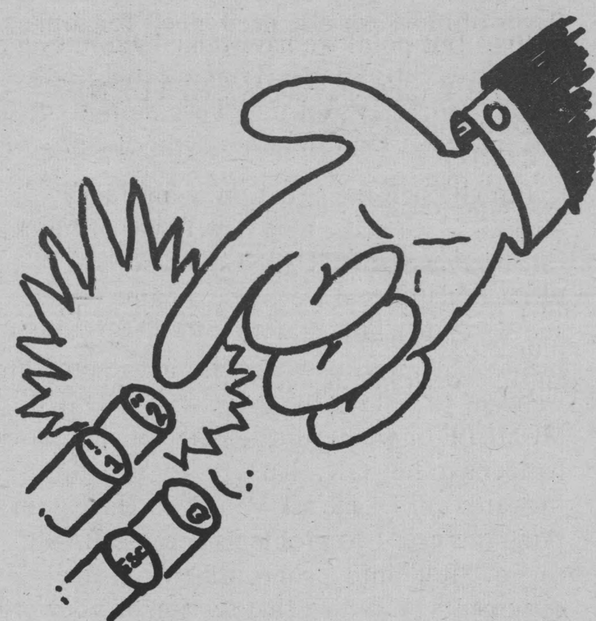


There is still much that computer science has to learn about control structures, and much fascinating and controversial work is being done in this field. BASIC sidesteps the issue entirely by being as bad as possible. Heaven protect me for saying this but even FORTRAN had argument passing. I'm not quite finished with DEF and we haven't even started on DIM but we're almost done with control structures so I will dispense with the last item on our list – GO TO.

Fairly recently it has been demonstrated that in languages with adequate control mechanisms GO TO's are unnecessary. In addition they have been shown to have various nasty side effects, so they have been blacklisted in computer science. They certainly can make a horrible mess out of an otherwise reasonable program. Here are two things you can do in your spare time to earn big money and learn why GO TO's are bad news. First, RUN the following program:

```
10 REM *** LAFF RIOT PROGRAM ***
20 LET N=1
30 FOR I=1 TO 2
40 GOTO 60
50 NEXT I
60 PRINT N;
70 LET N=N+1
80 GOTO 30
90 END
```

Either BASIC will blow up or go on PRINTing numbers forever. If your BASIC system blew up I will explain why a littler later. I also respectfully advise you not to RUN it if there are other users on the system as it might interfere with what they are doing if it blows up - - depending on the implementation. Unless of course, you're some kind of revolutionary or are in a nasty mood.



The second thing is a game you can play with your friends – I call it GO TO NUTS. Each player takes a simple program, say ten lines or so. Then rearrange the lines so their order makes no sense at all. Next insert a whole lot of GO TO's so that the program will RUN. Try to at least triple the length of the original program. Be sneaky, make long chains of GO TO's and put in groups of GO TO's that don't have anything to do with the program. Have more than one GO TO going to the non-GO TO lines. Players then trade programs. Winner is first person to figure out what the program he has is supposed to do. Booby prize goes to first player to go stark staring mad. Players caught in infinite loops are eliminated – RUNning the program is considered cheating. A bug in a program disqualifies the player who wrote it. Try it, you'll hate it, if you survive. Advanced users can no doubt invent more sophisticated variations.

Anyway, you get my point – keep the number of GO TO's in your program to an absolute minimum. BASIC sometimes forces you to use them, but avoid them whenever possible. Large programs with lots of GO TO's very often develop horrible bugs. The same sorts of problems can result from IF-THEN statements as well, because of computer science flaw number two.

Back to DEF. Information is passed to the one line "subroutine" in the form of arguments. Let's make up a "function" FNR so we will have something to point at.

```
10 DEF FNR(X,Y) = X – INT(X/Y)*Y
```

Now suppose that somewhere else in the program we were to write:

```
500 PRINT FNR(15,7)
```

When BASIC got to that line it would PRINT the number 1 (the remainder of 15 divided by 7). Now let's look a little more closely at what BASIC did. First it ignored the 500. Then it "saw" the word PRINT and got all ready to type something on the Teletype. Then it "saw" the label, FNR, of our subroutine. Then it passed the *value* 15 to the variable X, that is, it did a sort of "LET X=15" The same thing was also done for Y and 7. In computer science X and Y would be called *formal parameters* and 15 and 7 would be called *actual parameters.* The funny "LET" operation is called *binding.* The whole process looks like this in formal language: *Passing of arguments to subroutines is accomplished by binding the values of the actual parameters to the variables given as the formal parameters.*

Now suppose we changed our program a little so it looked like this:

```
500 LET X = 2
510 LET Y = 3
520 PRINT FNR(15,7),X,Y
```

When the computer got to that part of the program it would type out 1 2 3 (with a little more space between the numbers). But wait a minute; if BASIC *really* was binding 15 to X and 7 to Y it should type 1 15 7. Aha! We have caught the computer doing something "funny." The X in the DEF "block" is somehow different than the other X. They look the same, but variables given as formal parameters get special treatment *inside* the "block" they are used in. They are said to be *local* to that block. They are also sometimes called "dummy variables" but that is not a nice thing to call anybody. The idea of the same variable having different meanings or values in different parts of the program is called *scoping.* Now suppose we changed FNR so that instead of being defined in terms of X and Y we define it in terms of, say, P and Q. It would work just as well. So we can see that scoping is a very useful thing to have, since it allows us to define things without having to worry about what the actual parameters will be, in exactly the same way that variables themselves are useful because we can define things without having to worry about what their actual values are going to be. Now suppose we were to define another function, FNZ:

```
20 DEF FNZ(X,Y) = X + Y + Z
```

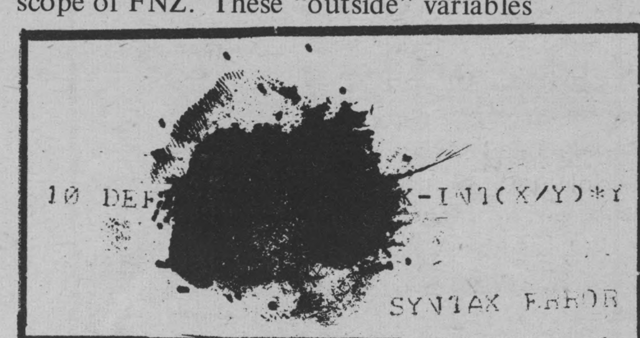
And changed the program so it included the following lines:

```
600 LET Z = 3
610 PRINT FNZ(1,2)
```

When it got to that part the computer would type a 6. Now let's change *just* Line 600 so that it looks like this:

```
600 LET Z = 4
```

Now the computer would type a 7. It is easy to see why this happens even though we did not change Line 610 – it is because we changed the value of Z. The computer does not care about the values we gave X and Y earlier in the program because they are local to FNZ. But it *did* matter what the value of Z was at that point because it was "outside" of the scope of FNZ. These "outside" variables



are said to be *global* (in this case Z is global with respect to FNZ). Now we can see another useful property of scoping; the various parts of our program do not have to "worry" about what "names" they give the variables they use unless they want to refer to *the same thing,* in which case that thing is global with respect to them. Exactly the same thing is true of people – it does not matter a bit if both of us have a secret name for something or other, such as gizmo, which we use when we are thinking or muttering to ourselves, but if we are out in the forest together and you say the word *bear* because one is standing behind me, I had better not think you are talking about the weather, because if I do I will get eaten up! So pardon the anthropomorphism of the last paragraph. To get back to the subject – BASIC only performs scoping inside of DEF blocks. Everywhere else, variables are *always* global. If people were like that it would be like hearing what everybody else was thinking, which would get pretty noisy. Another thing that happens because of this is that you have to be very careful that various parts of

your program do not clobber the value of a variable that another part of the program is using. Again, if people were like that and you were thinking about something and *anybody* anywhere else in the world (globe) used the word for it and changed its meaning the meaning of that word would also all of a sudden change inside *your* head, and what's worse, you'd never notice it until you started thinking nonsense, and maybe not even then. So here is computer science flaw number five:

BASIC does not provide the user with scoping. As a result he has to do a lot of bookkeeping. In addition, programs written in BASIC are liable to have bugs caused by conflicts in variable usage in different parts of the program.

Now what in the world does DIM have to do with all this? In answering this question I will fulfill the promise I made way back when, to explain what a language can be "about" besides numbers. To do this we will have to explore the idea of *value* a little more deeply.

When I used the word value in the previous sections chances are that you just thought "number" when you read it, and maybe thought I was trying to be a smarty pants by not just saying number right out. An example will straighten this out: If I asked you "what is the value of learning about computer science" and you said "two thousand eight hundred fourteen and a half" I would start to worry that you might get violent. I am not just pulling a semantic trick here – variables are just *names* for things, not necessarily the *same* thing all the time, but at any given instant they are the name of some-*thing*, just like the word "that" doesn't always refer to the same thing, but when I use it you know what I'm talking about, or you say "what are you talking about?" if you don't. The "thing" that the variable happens to be the "name for" at that moment is its value. Special names which always mean the same thing (have the same value) are called *constants.* For instance, 3 is the name for the "idea of three-ness" if people use it, but to a computer it is the name for a certain pattern of on and off bits.

So variables are general purpose names, just like "that." Now names can be used for whatever things we want to talk about, they're not picky. In computer languages what we want to talk about are three kinds of things: particular things or objects we are interested in for which we use *symbols* (because computers are dopes and don't know about anything else); patterns or systems that these things can form, which are called *structures;* and the ways in which these patterns can change, which are called *processes,* and are represented by programs.

Up to this point we have talked about everything except structures. True, we did discuss control structures, but they have more to do with processes than things or symbols. If you like you may call "thing patterns" *data structures* which is what is done in computer science. There is also a formal theoretical relationship between structures and processes, but I am not going to confuse the issue by going into all that. If you're interested, think of programs as patterns of symbols and see where it leads, or look into LISP which does away with the distinction almost entirely.

In BASIC, we have symbols for three things: numbers, strings of characters and variables. It is very useful to have symbols that are just plain symbols, but I would call this a defect not a flaw. BASIC provides only one type of structure for these objects – arrays. That is where the DIM statement comes in. The DIM statement is a member of a class of things computer scientists call *declarations.* A declaration is a statement which says "within the current scope these symbols represent objects which have this particular structure." In BASIC the particular structure indicated by the DIM statement is that of an "n-dimensional dense array with orthogonal euclidean connectivity." I give it's full name to show just how special a case it is. There are a whole lot of useful structures besides array thought, such as trees, directed graphs, sparse arrays, and sets. I will not go into what all these things are used for, I will only mention that inside the computer BASIC spends most of it's time doing things with data structures and only a small portion doing things with numbers. We are now ready to state computer science flaw number six:

BASIC provides only one data structure and contains absolutely no facilities for manipulating data structures. Since structures are one of the three main components of computation this flaw causes BASIC to be severely limited.

Besides the DIM statement there are three "hidden" or implicit declarations in BASIC – string variables are declared by the $ in their name, formal parameters are declared by their appearance inside the DEF statement, and the variable used in a FOR-NEXT loop is implicitly declared to be what is called *index variable.* In general, when a declaration is executed, a copy of the given data structure is created. These copies are sometimes called *incarnations.* That is why you can use expressions like

```
SIN(SIN(SIN(X)))
```

Each time the SIN routine in BASIC is called, a new copy of the formal parameter it uses is created. The LAUGH RIOT PROGRAM is intended to test whether the BASIC you're using creates copies of index variables. If it does, the program will fill up the computer's memory with copies of I (the index variable) and then blow up when the memory is full. The last number the program PRINTS will be a rough indication of how much memory you are given by your system. If memory is shared by the other users, the program will blow them up too (which isn't very nice).
If your BASIC does not create copies of index variables, the program will just RUN forever, and so much for that. Incarnation is a sort of "time scoping" and produces the same sort of advantages as regular scoping. Things like recursive functions are not possible without creating incarnations.

In conclusion –

BASIC has its uses, and can often be made to perform many of the functions I have outlined, but only if the user is willing to expend the necessary time and effort. It's not the worst introduction to computing, but you can only go so far before it's time to move on to better languages.

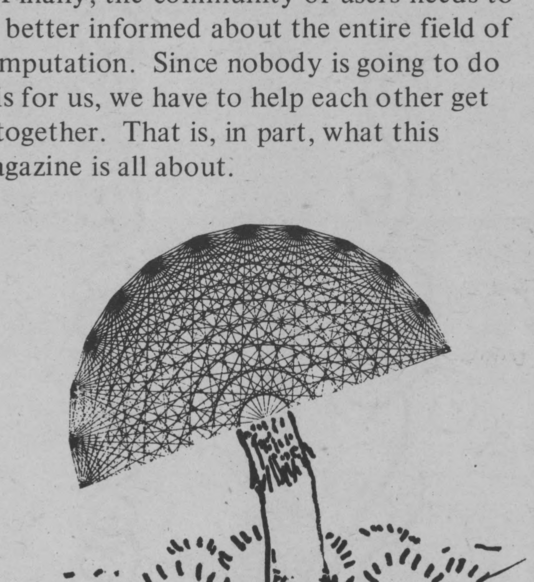
BASIC is workable but it is a whole lot less than what the user should be satisfied with. Manufacturers will continue to push BASIC until YOU, the consumer, DEMAND more and better quality languages and software. Any computer large enough for BASIC is also large enough for an ALGOL or LISP of equivalent complexity, and the software *may already exist*, but is not being pushed because, at the present time, BASIC is the big money maker.

Even ALGOL and LISP were long ago superceded by better languages, but commercial availibility lags so far behind development that there would be little point in my even mentioning them in this article. If you're interested, look into the PLANAR language, developed at MIT, which is itself becoming obsolete.

Another factor is that the educational/recreational market was really "cracked open" by BASIC, and the manufacturers will milk it for all it's worth before they introduce something else. And, they are not about to produce good instructional materials for other languages until then either. I'm not criticizing, that's just how it is right now. And the only way it's going to be any different is if YOU stop accepting whatever you're given and start DEMANDING what you need and want.

Furthermore, until now, there has never been any reason to develop languages and systems specifically for educational/recreational use. Here at PCC we are constructing a model of the "ideal" educational/recreational language, and you can help by sending us ideas and suggestions. Until the community of users gets a clear idea of what it wants and needs, we will not be able to supply manufacturers with constructive criticisms. The model is to help us do this.

Finally, the community of users needs to be better informed about the entire field of computation. Since nobody is going to do this for us, we have to help each other get it together. That is, in part, what this magazine is all about.



This article was written under extreme time pressure, and is not as good as it should be. Become an associate editor of PCC – rip out pages, scribble corrections and comments on them, or send us your questions, confusions and gripes. Next issue we will use your feedback to provide further inputs to the educational community. Maybe even a "User's Bill of Rights." Now it's YOUR turn. . .

This 1973 calendar was programed in Basic Edusystem 10

AREA CODE 415 392-6625 611

JANUARY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

FEBRUARY

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

MARCH

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

APRIL

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

JUNE

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

BY JA

Write a caler
Do one that
for a year, o
DECADE
cycle you ru

Then RUN
Watch the com
the days on l
paper.

Decorate you
watercolor p
borders arounc

on a little PDP8 computer with 4K memory & one teletype

NE WOOD

ndar program!
runs for a month,
r for a WHOLE
whatever time
n yourself on.

the program!
puter ZAP out
ong white teletype

calendar with
int or draw fancy
d the sides.

JULY

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

AUGUST

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

SEPTEMBER

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

OCTOBER

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

NOVEMBER

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

DECEMBER

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Terminal Terminology by *Roy Worthington*

President, Data Terminals & Communications

Computer Aided Instruction (CAI) and Timeshare systems for the most part use Teletype terminals as user input/output devices. The Model 33 series produced by Teletype Corp. in Skokie, Illinois has long been a standard and workhorse of the industry. While other terminals are more and more prevalent the economics and reliability of the Teletype are hard to beat.

Basically the 33 comes in two main varieties – ASRs and KSRs. The KSR (keyboard send/receive) is simply a keyboard and printing mechanism with an electronics (or call) control package for DAta Communication purposes. ASRs (automatic send/receive) have in addition, a paper tape reader and paper tape punch. Within both groups there are numerous options such as: sprocket (or pin) feed, automatic tape readers, automatic punches, various keyboard options etc. The price of a KSR is approximately two thirds the cost of an ASR. In considering an installation with several terminals in one location, a mix of KSRs and ASRs will provide cost savings against obtaining all ASR machines.

Most Timeshare and CAI system planners emphasize heavily on the types of computers and software available. Less thought goes into how users are going to "talk" to the system. Typically the attitude is: "we'll use a bunch of Teletypes which cost about $70 each a month" either being forgetful or unaware that a lot more is involved in connecting a terminal to a computer.

While each individual installation is unique in its terminal and data communication requirements, one or more of three main methods of connection are used.

Hardwired Terminals (Figure 1)

These are the simplest and generally lowest cost installation. Each port or channel to the computer is physically wired to the terminal (in the case of a Teletype a low cost interface box is required for compatibility—see Figure 1). Limitations of this type of installation are:

- Only good to 2000 feet maximum
- Each port is "tied" to a particular terminal making port and terminal inflexible

Advantages:

- Low installation cost – $35 for 50 feet of cable
- Low recurring cost – about $55 per month lease for an ASR 33 Teletype and interface
- No reliance on telephone lines and their inherent problems

"Dial Up" Terminals (Figure 2)

Probably the most expensive but definitely the most flexible method of coupling terminals to computers. Modems, which convert signals from the computer into audio tones for transmission over telephone lines and vice versa, are attached to the computer. These modems which connect to the phone line through a protective device called a Data Access Arrangement (DAA) are dialed by a user with a terminal and a coupler (the coupler is a modem used by the terminal and has a receptacle for placing a telephone hand set into for connecting to the phone line). Disadvantages:

- Higher cost – typically $65 per month for the terminal and $20+ for the modem at the computer
- Subject to phone line problems
- User not guaranteed a port should someone else occupy the line

Advantages:

- More terminals than lines can be provided. Many timeshare companies have five to eight times the number of customers than they could possibly handle at one time.
- Terminal not "tied" to one line should that line be down
- Other computers and even terminal to terminal communication can be used

Multiple Terminals Per Line (Figure 3)

The speed of a Teletype (10 characters per second – 110 baud) is low compared to the capacity of a phone line. A number of special modems or multiplexers are available that allow six or even eight terminals to use one phone line. In situations where four or more terminals are located some miles from the computer this type of hook up is ideal.

Disadvantages:

- Terminals are "tied" to a particular port
- Should the phone line go down all terminals on that line are out of service

Advantages:

- Phone line costs reduced significantly
- Prevents phone line being misused (it only goes to the computer)

Should you require further information –

Data Terminals & Communications
535 Salmar Avenue
Campbell, California 95008

will be pleased to answer your questions and for a nominal charge will draw up a specific system for your particular applications.

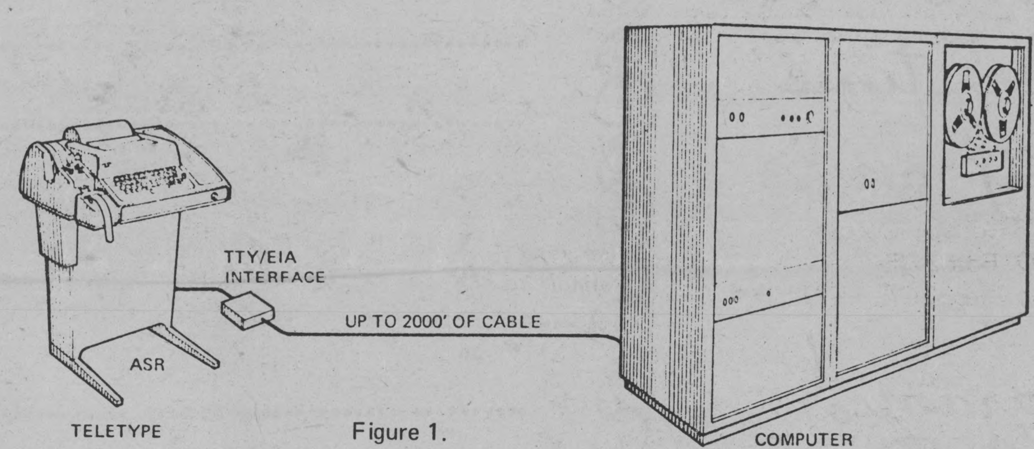


Figure 1.

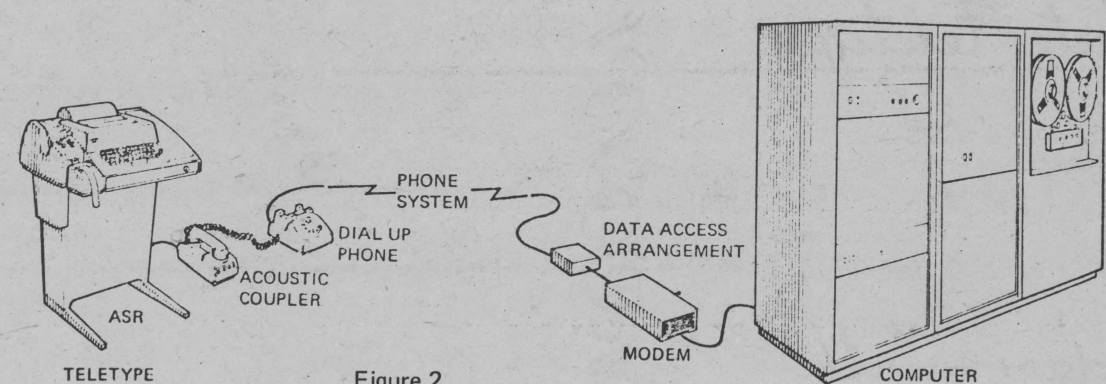


Figure 2

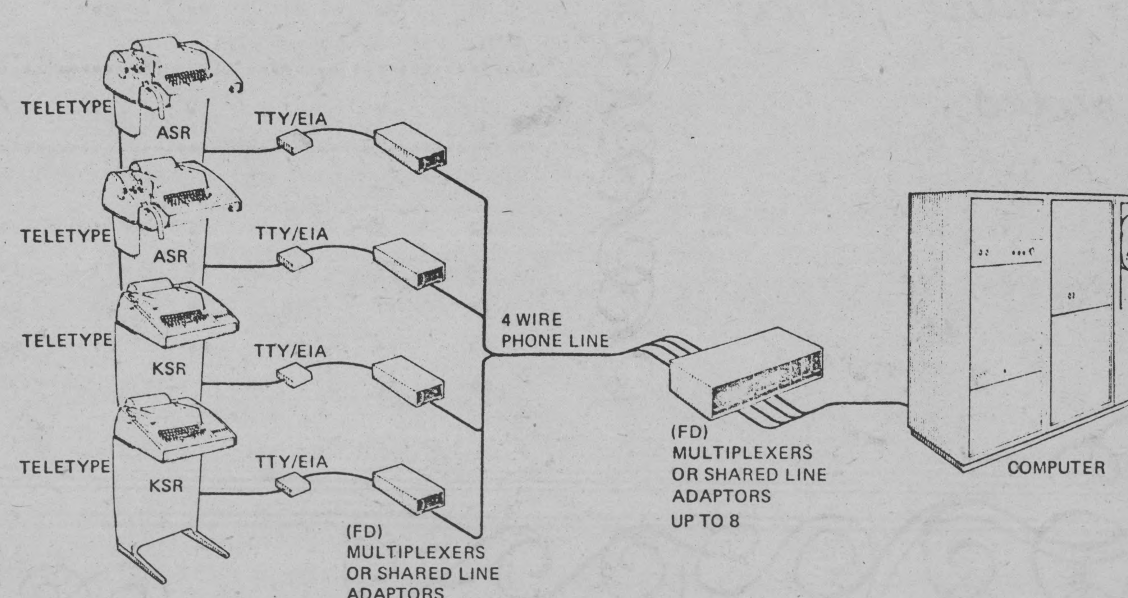
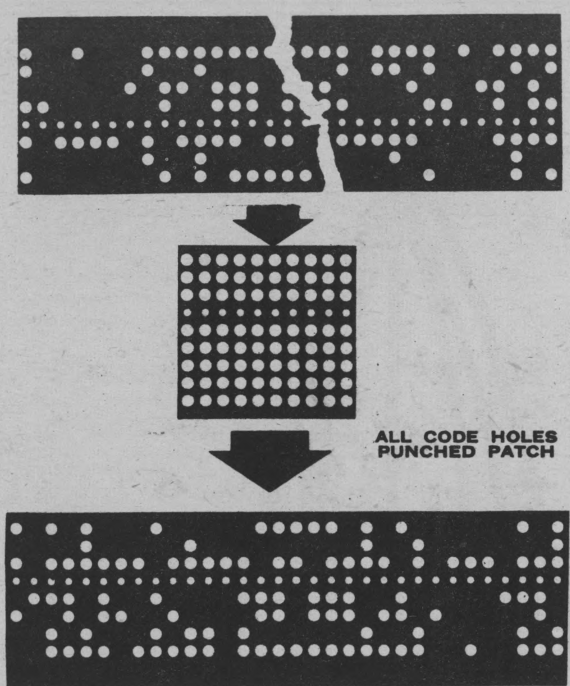


Figure 3

DATA–LINK

Ever tear a paper tape? Ever tear the last copy of a long tape? Good news! You can buy patches to quickly repair those torn tapes.



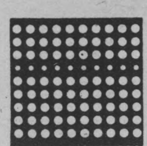
TORN TAPE

\+

PATCH

=

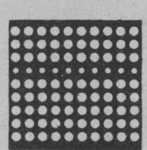
ALMOST AS GOOD AS NEW!



MYLAR OPAQUE

MYLAR® OPAQUE PATCHES — are the thinnest patches available and are the perfect answer to photo electric readers . . . eliminates errors from light emission. There are 1,000 1-inch patches with peel-off backs per Kit with all code and feed holes punched.

$20 PER THOUSAND



PAPER PRESSURE SENSITIVE

PAPER PRESSURE SENSITIVE PATCHES — are extra strong and are especially suited for use with oiled tapes. Peel off back makes them easy and fast to apply. There are 1,000 1-inch patches per Kit, with all code and feed holes punched.

$19 PER THOUSAND

from: DATA–LINK CORPORATION
733 Convoy Court
P.O. Box 2729
San Diego, CA. 92112

Ask them for a free sample kit

What do you want to know about TTYs?

DTC

Data Terminals and Communications maintains more than 200 Teletypes in the San Francisco Bay Area. 75% of these are in educational places. DTC guarentees that they will appear on site to fix your TTY within 8 hours after a service call and they claim and average response time of 4 hours. If they have to take your sick TTY to the shop, they'll leave you a loaner.

CHARGES

ON CALL SERVICE:	$9.50 per hour fro ASR 33
CONTRACT:	$15 to $19 per month for ASR 33 $3 per month for acoustic coupler

DTC will also rent, lease or sell you TTYs, interfaces, acoustic couplers and modems — new or used.

SAMPLE PRICES

	Monthly Rental*	Yearly Lease*	Purchase
ASR 33 (new)	$50	$49	$1100
ASR 33 (rebuilt)	–	–	$ 825
TTY to EIA interface	$14	$ 4	$ 50
Prentice DC22 acoustic coupler	$15	$14	$ 298

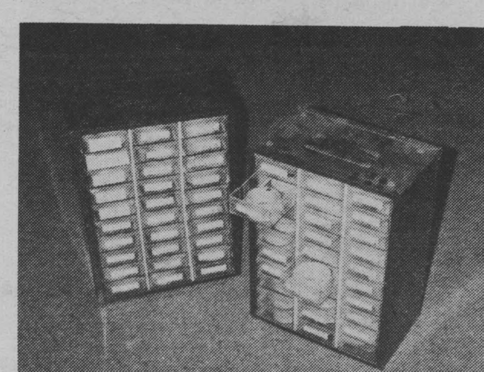
* Includes maintenance

from: Data Terminals & Communications
535 Salmar Avenue
Compbell, California 95008

HOW DO YOU STORE TAPES?

We used to store program on paper tape like this:

But now we store them like this. We got our storage cabinet at Sears (30 drawers, about $7.50). Watch for sales at your local discount house.



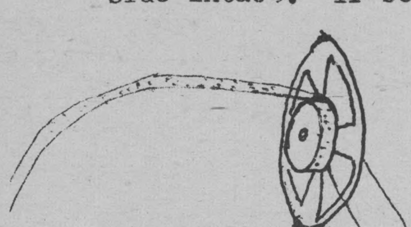
How do you store tapes?

CHEEP TAPE WINDER:

??? 1 400' (7" dia) metal film take-up reel
1 16mm movie projector (any model)
1 take-up reel demolisher (look in your toolkit--you probably thought it was a pair of pliers or a hacksaw)

Demolish the takeup reel by removing one side of the takeup reel down to, but not including, the hub. If the reel has differently shaped center holes, remove the side that has the ROUND hole, leaving the square hole side intact. If both sides are the same, either one will do. A LOT OF REELS WON'T WORK FOR THIS BECAUSE OF THEIR CONSTRUCTION, BUT KEEP LOOKING.

(Of course if you don't have a 16mm projector, it isn't so cheap! – PCC)

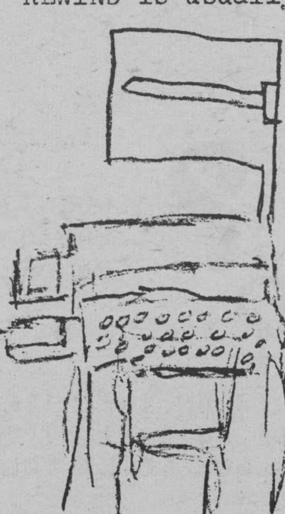


Mount the reel on the movie projector, mount the tape on the reel like a film, turn the projector to RUN (without the lamp), turn off when tape is wound. REWIND is usually too fast.

Keep up the good work.

Paul

Paul W. Marsh
Box 15350 WEDGEWOOD
SEATTLE 98115



~~TELEPY~~TYPE IS NEET ~~BU~~ AND ~~I~~ DONT THINK THERE IS ANYTHING ~~I~~ DONT LIKE ABOUT IT.

BASIC

MUSIC IN BASIC?

Why not?! We don't know of many BASIC systems that can *play* music directly, like the big computer music setups, but with humans to do the planning, programming, and playing, the small computer becomes a useful tool for people who know music or want to learn. There are even some published music programs in BASIC (we'll try to review these in later issues of PCC). The programs in this issue were concocted by local music freaks, however.

ØVER

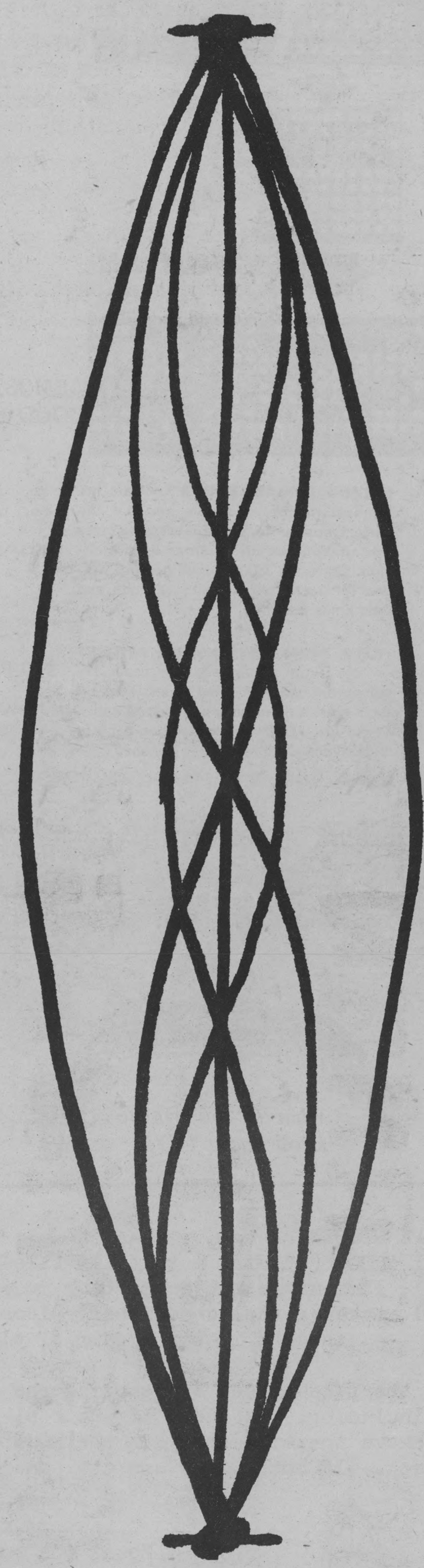
GOOD VIBRATIONS

To start with, let's look at musical pitch. Sound is made by something vibrating, i.e., moving back and forth in some regular way. The faster the rate of vibration (i.e., the *frequency*) the higher the pitch of the sound.

Do you have a guitar? Try this experiment.

First, strike any string so that it makes a good loud sound. Then, place your finger gently on the string right over the 12th fret. DON'T PRESS THE STRING DOWN TO THE FINGERBOARD, just TOUCH IT.

The result should be a bell-like ringing sound higher in pitch than the sound of the open string.



TØNE

THE OVERTONE (OR HARMONIC) SERIES

This ringing sound is called an overtone. The overtone you heard is produced by the string vibrating in parts (like the diagram along the side of the page yonder). In this case the string is vibrating in halves (the 12th fret is exactly halfway between the bridge and the nut). Any integer division of the length of the string will produce an overtone. The number of overtones is theoretically infinite, since the integers go up to infinity. However, the higher ones are hard to locate or hear. The first eight overtones (at 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, and 1/8 of the string length) are pretty audible.

Also, the string vibrating as a whole (1/1) is an overtone! Now ... experiment some more ... try other places on the string. Can you get any more overtones?

SERIES

QUESTIØN: ARE ALL THE ØVERTØNES YOU ØBTAINED LØCATED RIGHT ØVER FRETS ØN THE GUITAR FINGERBØARD? ARE THERE ANY THAT SEEM TØ BE BETWEEN FRETS? HØW IS THE GUITAR FINGERBØARD CØNSTRUCTED?

QUESTIØN: IS THERE ANY PHYSICAL BASIS FØR THE CØUNTING NUMBERS, ØR ARE THEY JUST IMAGINARY CØNCEPTS?

QUESTIØN: IF THERE IS AN INFINITE NUMBER ØF ØVERTØNES, AM I RIGHT TØ SAY, "WHEN YØU HEAR ØNE NØTE, YØU HEAR THEM ALL"?

MUSIC

??? ?

INTERVALS

One more idea, before we look at some programs. The musical interval between tones is determined by the ratio of the frequencies of the tones. For example, if one tone has a frequency of 100 vibrations per second, and another has a frequency of 300 v/sec, then the ratio of their frequencies is 3/1. This idea about intervals and frequencies was worked out by Pythagoras, as was the idea of overtones as integer multiples of a fundamental frequency (the fundamental is the 1/1 overtone frequency).

Other peoples have recognized this principle, however, and there are elaborate Chinese, Arabic, etc., music theories based on this definition of intervals. The Chinese may have learned of these concepts from travelers who knew about Pythagoras.

```
QUESTIØN: IS THERE ANY CØNNECTIØN BETWEEN MUSIC
AND RIGHT TRIANGLES?  WHAT INTERVAL IS GIVEN BY
THE RATIØ 3/4?  4/5?  3:4:5?
```

OCTAVES

One of the most important intervals worldwide is the ratio 2/1. Western (i.e., European, etc.) musicians call this interval the octave. There is a tendency in human musical culture to use octaves as "defining points" in making up musical patterns. Most scales essentially begin repeating every octave. Why is this?

The following BASIC program computes the overtones of an INPUT fundamental (F). The program also computes the ratio, and reduces it to a decimal that lies between 1 and 2. Since any power of 2/1 is also an octave of the fundamental (an octave of an octave is an octave), all octaves of F can be said to have ratio 2/1 without falsifying the relationship. Reducing the ratio is done in lines 190 to 210. Here is the program, and a RUN for the first 10 overtones.

```
100  REM ØVERTØNE SERIES GENERATØR
110  PRINT "HØW MANY ØVERTØNES DØ YØU WANT";
120  INPUT T
130  PRINT "WHAT IS THE FUNDAMENTAL FREQUENCY";
140  INPUT F
150  PRINT
160  PRINT "ØVERTØNE","FREQUENCY","RATIØ"
170  FØR I=1 TØ T
180  H=F*I
190  R=I
200  IF R <= 2 THEN 230
210  R=R/2
220  GØTØ 200
230  PRINT I,H,R
240  NEXT I
250  PRINT
260  END
RUN

HØW MANY ØVERTØNES DØ YØU WANT?10
WHAT IS THE FUNDAMENTAL FREQUENCY?440

ØVERTØNE        FREQUENCY       RATIØ
 1                440            1
 2                880            2
 3                1320           1.5
 4                1760           2
 5                2200           1.25
 6                2640           1.5
 7                3080           1.75
 8                3520           2
 9                3960           1.125
 10               4400           1.25
```

F IS FØR FUNDAMENTAL.

H IS FØR HARMØNIC.

R IS FØR RATIØ.

P IS FØR PYTHAGØRAS.

!!!!

```
QUESTIØN: HØW MANY ØCTAVES ARE THERE IN THE FIRST
10 ØVERTØNES?  ARE THERE ANY ØTHER RATIØS ØCCURRING
AS ØFTEN IN THE FIRST 10?  DØES YØUR EAR PRØDUCE
ØVERTØNES WHEN YØU HEAR SØUNDS?  WHY MIGHT THE
ØCTAVE BE "UNIVERSALLY" IMPØRTANT TØ HUMANS?
```

SCALES

The number of possible frequencies is infinite, because the spectrum of sound is continuous. In practice, human musicians have attempted to select definite frequencies to work magic/music with. A set of definite tones (or intervals, which amounts to the same thing) is called a scale.

There are infinitely many different scales. Pythagroas constructed a 7-tone scale based on the 3/1 overtone (next most important after the octave). The Pythagorean scale contains tones at intervals of 3/2, starting with a tone 2/3 that of the Fundamental. Try to write a program to compute the frequencies of such a scale. We'll publish our version next issue.

Meanwhile, on to the 18th Century and J.S. Bach. The scales of Pythagoras and other variations (we'll get to them next time too) implied a scale of 12 tones to the octave, increasing proportionately (i.e., geometrically) from F to 2*F. Scales of this type are called tempered scales, and were apparently known as theoretical possibilities to the ancient Greeks. Only by the time of Bach, however, was the 12 tone tempered scale a practical possibility. Why was it so difficult? Because the proportionality constant for a 12 to the octave scale has to be the twelfth root of two!!! Think about it ... the first tone has frequency F, the second has F*C, the third has F*C*C of $F{*}C^2$. Tone 13 must have frequency $F{*}C^{12}$, and must also equal 2*F, since it represents the octave (12 different tones, remember?).

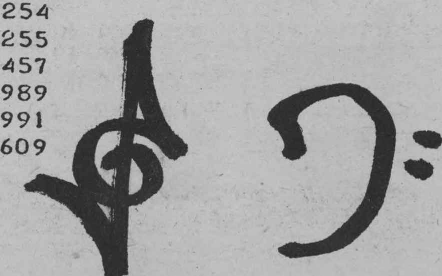
If $F{*}C^{12} = 2{*}F$, then $C^{12} = 2$ and $C = 2^{1/12}$, the 12th root of 2.

This is an irrational number (like all roots of 2), and is difficult to calculate without logarithms. Logarithms were discovered by Napier in the late 16th Century (see Newman, *The World of Mathematics, Vol. 1*, p. 123), and tables of logarithms were probably not generally available (or appreciated) until nearly a century later. So, Bach's "Well-Tempered Clavier" could have been written earlier, but could not have been played at one sitting.

The program opposite calculates the frequencies of a tempered scale with an INPUT number of tones in one octave. It uses the LOG and EXP functions; we could have used an expression like $F{*}2^{1/12}$ instead, but this way seems neat. No doubt Napier, after 25 years of hand calculations to produce the first book of log tables, would rather have done it this way!

BASIC

```
100  REM *** SCALE FREQUENCY GENERATØR ***
110  PRINT
120  PRINT "HØW MANY TØNES TØ THE ØCTAVE ";
130  INPUT T
140  PRINT
150  PRINT "WHAT IS THE BASE FREQUENCY ";
160  INPUT F
170  PRINT
180  LET L=LØG(2)/T
190  PRINT "TØNE","FREQUENCY"
200  PRINT
210  FØR I=0 TØ T-1
220  PRINT I+1,F*EXP(I*L)
230  NEXT I
240  PRINT
250  END

RUN


HØW MANY TØNES TØ THE ØCTAVE ?12

WHAT IS THE BASE FREQUENCY ?440

TØNE          FREQUENCY

 1             440
 2             466.164
 3             493.883
 4             523.251
 5             554.365
 6             587.329
 7             622.254
 8             659.255
 9             698.457
 10            739.989
 11            783.991
 12            830.609
```



TEMPER, TEMPER

Earlier we discussed various properties of single tones, such as their overtones. The program at the left, titled "BEAT FREQUENCY ANALYZER" allows us to explore, in a simplified way, one of the musical effects produced when two tones are sounded together. This effect is called *beating* and is one of the main factors which determine whether a pair of tones are consonant (or dissonant) with respect to each other when they are played together. Whenever two frequencies are vibrating simutaneously a complicated motion is set up in the vibrating "medium" or substance, since it is being made to vibrate in two different ways at once. One of the effects caused by this compound vibration is that, in addition to hearing the two tones being played, we also hear *another tone* whose frequency is *the difference between the frequencies of the other two tones.* This sound is called the *difference tone.* Because of the way our ears work there is a lower limit to the frequencies we perceive as continuous sounds. Below this limit frequencies are heard as separate "pulses" or beats. This lower limit varies from person to person and is also somewhat dependent upon other factors as well, but for our purposes we will say it lies somewhere between 20 and 60 cps (cycles per second) say something like 30 cps. If we examine still lower frequencies, below 10 cps for instance, we find that the beats are coming too slowly for us to notice, and we usually don't hear them at all. Now, if the difference tone has a frequency in this range (10 to 30 cps) it sort of "breaks up" the sound we are hearing in what is usually considered an "unpleasant" or dissonant manner. We might say that it is "sour" or that the tones are "out of tune." Since most musical tones have overtones these also produce difference tones with *all the overtones of the other sound* as well (what are the difference tones produced by the overtones of a *single* tone?) and if any of these difference tones lie in the critical range the same sort of effect occurs, they sound "dissonant."

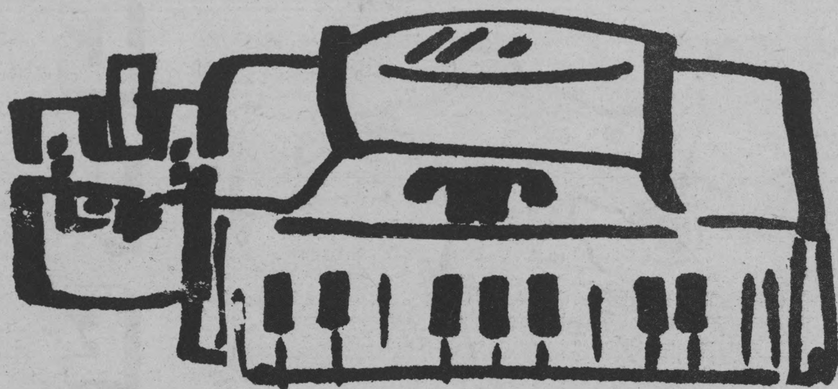
At this point we have enough background information to discuss the program. As the RUN here indicates we are first asked to INPUT the "base frequency." This is the frequency from which the frequencies of the 12 tone scale are computed. We have used the conventional "A above middle C" or 440 cps. Then the program asks for the "upper" (30 cps) and "lower" (10 cps) limits of the critical range. These are INPUT so the effects of changing the limits can be explored.

Next the program asks for the two chromatic scale tones you wish to examine. The base frequency is considered to be zero, positive numbers represent tones above the base frequency in pitch, negative numbers represent tones below the base frequency. With the base frequency of 440 cps as in our example, 12 would represent 880 cps and −9 would be the number to INPUT for "middle C."

```
100  REM *** HARMØNIC BEAT-FREQUENCY ANALYZER ***
110  PRINT
120  L=(LØG(2))/12
130  PRINT "INPUT BASE FREQUENCY, LØWER LIMIT, UPPER LIMIT ";
140  INPUT F,D0,D1
150  DEF FNT(X)=F*EXP(X*L)
160  PRINT
170  PRINT "INPUT FIRST TØNE, SECØND TØNE ";
180  INPUT P,Q
190  PRINT
200  U=FNT(P)
210  V=FNT(Q)
220  K=1
230  T=K*V
240  J=1
250  S=J*U
260  W=ABS(T-S)
270  IF W>D1 THEN 310
280  IF W >= D0 THEN 350
290  PRINT "CØNSØNANT ";
300  GØTØ 360
310  J=J+1
320  IF S<T THEN 250
330  K=K+1
340  GØTØ 230
350  PRINT "HARMØNIC #1","HARMØNIC #2","BEAT FREQUENCY"
360  PRINT J,K,W
370  GØTØ 160
380  END
RUN


INPUT BASE FREQUENCY, LØWER LIMIT, UPPER LIMIT ?440,10,30

INPUT FIRST TØNE, SECØND TØNE ?1,8


CØNSØNANT  3                    2                    1.57812

INPUT FIRST TØNE, SECØND TØNE ?1,5

HARMØNIC #1     HARMØNIC #2     BEAT FREQUENCY
 5                4               18.499
```

(Continued on the top of the next page)

MUSIC



(Continued from page 16)

The program then computes the difference tone produced by these frequencies, and also the difference tones produced by their overtones until it finds one whose frequency is less than the frequency we INPUT as the "upper limit." It then checks to see whether the difference tone produced is in the critical range (i.e., its frequency is greater than the "lower" limit). If the tone is in the critical range it PRINTs out which overtone (also called *harmonic*) of each of the two original tones produced the difference tone, and also PRINTs the frequency of the difference tone (also called the beat frequency). If, on the other hand, the beat frequency is less than the number given as the "lower" limit of the critical range the two tones are considered "consonant" so it PRINTs the word CONSONANT followed by the three values mentioned above.

We mentioned previously that this program allows us to explore the phenomenon of beat frequencies in a simplified way. The perception of sound is a very complicated subject, and any very precise computer model that is to be used to explore the intricate ideas of "consonance" would have to take this into account.

In any case, despite its limitations, the program is a fairly good place to begin to get a feel for the subject, and hopefully will help you in your own experimenting.

10 PRINT "♫"

Now for a taste of things to come. In some subsequent issue of PCC we hope to review composing simulators in BASIC. For example, Kemeny and Kurtz have an interesting harmony generator which we'll try to review/reprint. As a starter, we present one of our efforts in the genre: the MAD COMPOSER. This composer can keep his tunes "in key" even if the key "spelling" isn't always correct (no flats – you may wish to modify the program to eliminate this deficiency). Line 430 selects a number at random, 1 through 7. Line 440 then gets a tone from the circle of fifths (the Chromatic scale arranged with a chromatic interval of 7 between adjacent tones). Since any 7 adjacent tones on the circle of fifths is a key, this step ensures that the correct key (INPUT by the user) will be used.

The program has string input and output; a sample RUN is reprinted with the program.

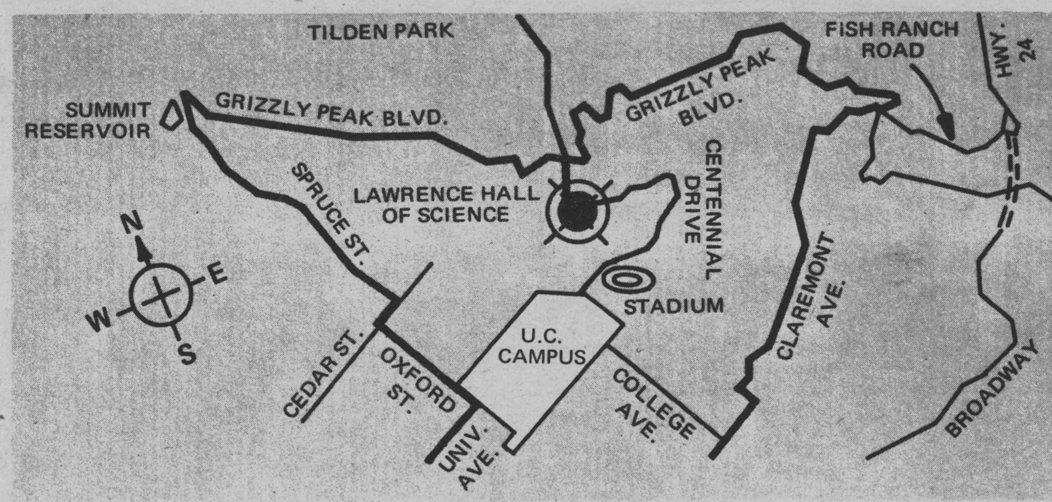
We hope that this program will stimulate some of you readers to write composers of your own. If you do, and you think we'll like them, send them in (attn: PLS). Also, feel free to use any of our music programs, for free if you don't charge for them yourself.

Written for HP 2000C

```
100  REM ***RANDOM MELODY GENERATOR WITH STRINGS***
110  REM ***COPYRIGHT 1972 BY DYMAX***
120  DIM M[100],A$[24],K$[2],J$[2],G$[3]
130  A$="A A#B C C#D D#E F F#G G#"
140  MAT M=ZER
150  PRINT "DO YOU WISH INSTRUCTIONS";
160  INPUT Q$
170  IF Q$="NO" THEN 260
180  PRINT "I AM A (SLIGHTLY) MAD COMPOSER.  MY MELODIES MAY"
190  PRINT "AMUSE YOU.  YOU TELL ME THE KEY, AND HOW MANY NOTES"
200  PRINT "TO COMPOSE, AND I WILL DO THE REST.  "
210  PRINT "THE KEY MAY BE A SINGLE LETTER, A B C D E F G, OR"
220  PRINT "IT MAY BE A LETTER FOLLOWED BY '#' (SHARP)."
230  PRINT "NO FLAT KEYS, PLEASE.  "
240  PRINT "FOR EXAMPLE, TYPE F FOR F-NATURAL, F# FOR F-SHARP."
250  PRINT "THE MELODY MAY BE UP TO 100 NOTES LONG."
260  PRINT "WHAT KEY DO YOU WISH";
270  INPUT K$
280  J$="  "
290  IF LEN(K$)=2 THEN 320
300  J$[1,1]=K$
310  K$=J$
320  FOR J=1 TO LEN(A$)-1 STEP 2
330  IF K$=A$[J,J+1] THEN 350
340  NEXT J
350  K=(J+1)/2
360  REM**K IS NOW A NUMBER, 1<=K<=12, REPRESENTING THE KEY.***
370  PRINT "HOW MANY NOTES DO YOU WISH";
380  INPUT N
390  PRINT
400  PRINT
410  REM ** THIS IS THE MELODY GENERATOR **
420  FOR I=1 TO N
430  Y=INT(7*RND(0))+1
440  M[I]=7*(Y-2)+K+12
450  REM **NOW TO REDUCE THE TONE MOD 12 **
460  M[I]=M[I]-INT(M[I]/12)*12
470  IF M[I]#0 THEN 490
480  M[I]=12
490  Q=2*M[I]-1
500  PRINT A$[Q,Q+1];" ";
510  NEXT I
520  PRINT
530  PRINT
540  PRINT
550  PRINT "HOW ABOUT THAT ONE?  WANT ANOTHER";
560  INPUT Q$
570  IF Q$="YES" THEN 260
580  PRINT
590  PRINT "THAT'S ALL, FOLKS !!"
600  END
RUN
MAD

DO YOU WISH INSTRUCTIONS?YES
I AM A (SLIGHTLY) MAD COMPOSER.  MY MELODIES MAY
AMUSE YOU.  YOU TELL ME THE KEY, AND HOW MANY NOTES
TO COMPOSE, AND I WILL DO THE REST.
THE KEY MAY BE A SINGLE LETTER, A B C D E F G, OR
IT MAY BE A LETTER FOLLOWED BY '#' (SHARP).
NO FLAT KEYS, PLEASE.
FOR EXAMPLE, TYPE F FOR F-NATURAL, F# FOR F-SHARP.
THE MELODY MAY BE UP TO 100 NOTES LONG.
WHAT KEY DO YOU WISH?A
HOW MANY NOTES DO YOU WISH?10


F#  D   D   F#  B   F#  G#  C#  D   A


HOW ABOUT THAT ONE?  WANT ANOTHER?YES
WHAT KEY DO YOU WISH?C
HOW MANY NOTES DO YOU WISH?10


E   B   A   A   A   G   B   E   A   B


HOW ABOUT THAT ONE?  WANT ANOTHER?NO

THAT'S ALL, FOLKS !!
```

Peter Lynn Sessions and friends at Peninsula School Learning Fair (see Saturday Review of Education, January 1973). Peter and Marc's Electric Bead Game Workshop is described on page 2 of this issue.

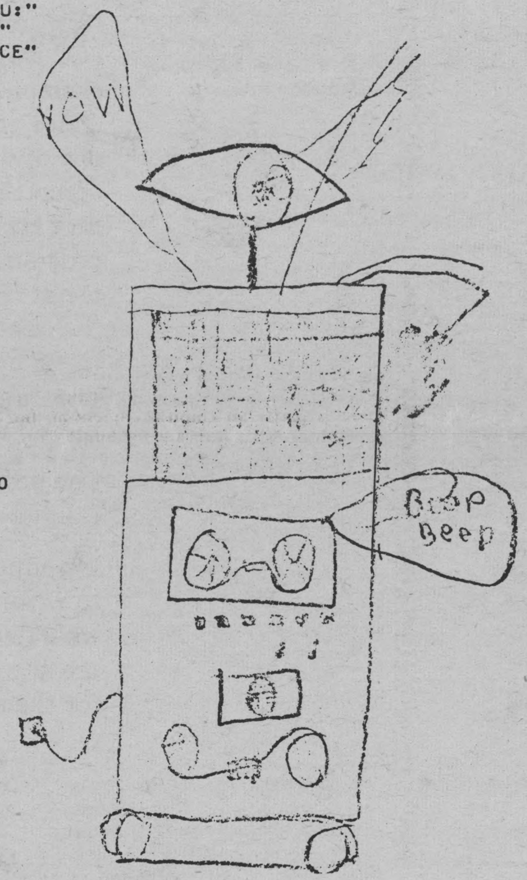
LAWRENCE HALL OF SCIENCE ~ BERKELEY



Beginning this issue, we will print listings of one or more Lawrence Hall of Science game playing programs (for HP 2000B) along with other news from the Hall. LHS is one of our favorite sources of computer games and one of our favorite games is BAGELS.

bagels

```
1000  REM **PICO-FERMI-BAGELS NUMBER GUESS GAME**  D. RESEK, P. ROWE
1010  REM COPYRIGHT 1971 BY THE REGENTS OF THE UNIV. OF CALIF.
1020  REM PRODUCED AT THE LAWRENCE HALL OF SCIENCE, BERKELEY
1030  DIM A$[10],N$[10],A[3],B[3]
1040  N$="0123456789"
1050  Y=0
1060  T=255
1070  PRINT "WOULD YOU LIKE THE RULES";
1080  INPUT A$
1090  IF A$[1,1]="N" THEN 1150
1100  PRINT "I AM THINKING OF A THREE DIGIT NUMBER.  YOU CAN GUESS WHAT"
1110  PRINT "NUMBER I HAVE IN MIND AND I WILL TELL YOU:"
1120  PRINT "PICO   - ONE DIGIT IS IN THE WRONG PLACE"
1130  PRINT "FERMI  - ONE DIGIT IS IN THE CORRECT PLACE"
1140  PRINT "BAGELS - NO DIGIT IS CORRECT"
1150  FOR I=1 TO 3
1160  A[I]=INT(10*RND(0))
1170  FOR J=1 TO I-1
1180  IF A[I]=A[J] THEN 1160
1190  NEXT J
1200  NEXT I
1210  PRINT "OKAY, I HAVE A NUMBER IN MIND."
1220  FOR I=1 TO 20
1230  PRINT "GUESS #"I":";
1240  ENTER T,C,A$
1250  IF LEN(A$)#3 THEN 1630
1260  FOR J=1 TO 3
1270  FOR C=1 TO 10
1280  IF A$[J,J]=N$[C,C] THEN 1320
1290  NEXT C
1300  PRINT TAB(22)"WHAT?";
1310  GOTO 1230
1320  B[J]=C-1
1330  NEXT J
1340  IF B[1]=B[2] OR B[2]=B[3] OR B[1]=B[3] THEN 1650
1350  C=D=0
1360  FOR J=1 TO 2
1370  IF A[J]#B[J+1] THEN 1390
1380  C=C+1
1390  IF A[J+1]#B[J] THEN 1410
1400  C=C+1
1410  NEXT J
1420  IF A[1]#B[3] THEN 1440
1430  C=C+1
1440  IF A[3]#B[1] THEN 1460
1450  C=C+1
1460  FOR J=1 TO 3
1470  IF A[J]#B[J] THEN 1490
1480  D=D+1
1490  NEXT J
1500  IF D=3 THEN 1680
1510  PRINT TAB(22);
1520  FOR J=1 TO C
1530  PRINT "PICO ";
1540  NEXT J
1550  FOR J=1 TO D
1560  PRINT "FERMI ";
1570  NEXT J
1580  IF C+D THEN 1600
1590  PRINT "BAGELS";
1600  NEXT I
1610  PRINT "OH WELL"
1620  GOTO 1700
1630  PRINT "TRY GUESSING A THREE DIGIT NUMBER."
1640  GOTO 1230
1650  PRINT "OH.  I FORGOT TO TELL YOU THAT THE NUMBER I HAVE IN MIND"
1660  PRINT "HAS NO TWO DIGITS THE SAME."
1670  GOTO 1230
1680  PRINT "YOU GOT IT"
1690  Y=Y+1
1700  PRINT "AGAIN";
1710  INPUT A$
1720  IF A$[1,1]="Y" THEN 1150
1730  IF Y=0 THEN 1750
1740  PRINT "A  "Y"- POINT BAGELS BUFF"
1750  END
```



A Big Box full of wires, meanory banks, panels, buttons, math problems

!!COMPUTER POWER!!

The computing capability of LHS took a giant step forward on November 2, with the delivery of a Hewlett-Packard System 3000. This, the first 3000 installation by HP, brings to four the number of computers at LHS, and adds 16 terminals to the Hall's existing capability for more than 80. However, the HP/3000, which its manufacturer calls a mini-computer, has many features which make it a maxi-mini:

It allows concurrent timesharing and batch operator.

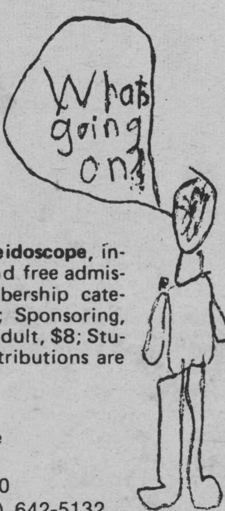
In either mode, users have access to three languages, BASIC, FORTRAN, and Systems Programming Language, as well as a Text Editor.

The operating system, called the *M*ulti-*P*rogramming *E*xecutive, manages all systems resources, assigning them to users as required.

A vertical memory with segmentation is utilized.

The system is to be used generally for research. The first such project scheduled is the implementation of the LOGO programming language. The instructional needs of LHS will continue to be filled by the DECISION and HP/2000B systems. The system configuration at the Hall includes 10 million bytes (characters) of disk storage and a 200-line-per-minute line printer.

Rumor has it that after the HP 3000 was delivered, the HP 2000B began crashing erratically. Insiders at LHS think the 2000 is jealous of the 3000.



We played bagels. I t was fun because I beat the computor three times out of five. I hope we play it again so I can beat it.

The computor is sorry at four dimensional tic tac toe

But it was fun!

ba

LOOKING OVER THE LITERATURE. jrb

The First Intergalactic Space War Olympics: Fanatic Life and Symbolic Death Among the Computer Bums, by Stewart Brand; *Rolling Stone*, Issue No. 123, December 7, 1972.

If you missed this delightful article, grab any passing long haired rock music freak and borrow a copy of the recent issue of Rolling Stone with Carlos Santana on the front. Stewart has rounded up interviews with many of the most creative people involved in the most interesting activities at the forefront of people's computer use and research. Here are a couple of paragraphs I especially liked from the article's conclusion:

Until computers come to the people we will have no real idea of their most natural functions. Up to the present their cost and size has kept them in the province of rich and powerful institutions, who, understandably, have developed them primarily as bookkeeping, sorting and control devices. The computers have been a priceless aid in keeping the lid on top-down organization. They are splendidly impressive as oracles of (programmable) Truth, the lofty voice of unchallengeable authority.

In fact, computers don't know shit. Their especial talent in the direction of intelligence is the ability to make elaborate models and fiddle with them, to answer in detail questions that begin "What if . . ?" In this they parallel (and can help) the acquiring of intelligence by children. But the basic fact of computer use is "Garbage In, Garbage Out"—if you feed the computer nonsense, it will dutifully convert your mistake into insanity-cubed and feed it back to you. Children are different—"Garbage In, Food Out" is common with them. Again, the benefits of variant parallel systems. Computer function is mostly one-track-mind, in which inconsistency is intolerable. The human mind functions on multiple tracks (not all of them accessible); it can tolerate and even thrive on inconsistency.

I suggest that the parallel holds for the overall picture of computer use. Where a few brilliantly stupid computers can wreak havoc, a host of modest computers (and some brilliant ones) serving innumerable individual purposes can be healthful, can repair havoc, feed life. (Likewise, 20 crummy speakers at once will give better sound fidelity than one excellent speaker—try it.)

PERIODICALS

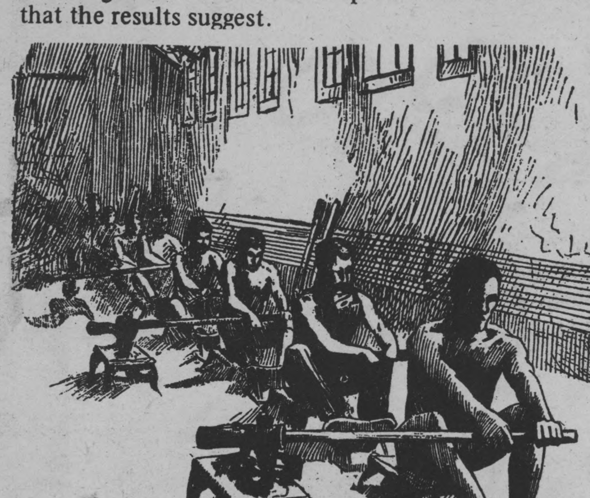
Simulation/ Gaming/News

S/G/N is our window looking in on the world of simulations, games and simulation games. Each issue has news, philosophy and a complete game you can play.

***Simulation Gaming News*, an Alohn and Hyer publication, is issued five times a year (every other month except in the summer). Subscriptions are $4 for five issues, and checks (not purchase orders) should accompany subscription requests. Zip codes also should be supplied. Advertising rates and specifications are available upon request. Communications should be addressed to S/G/N, Box 8899, Stanford University, Stanford, Ca. 94305.**

Since 1952 the Simulation Councils have promoted simulation. Now, by no means because of these efforts alone but rather because nothing is so powerful as an idea whose time has come, simulation has permeated all aspects of our lives. As a result, many people are now running simulations, and they and many others are "buying" the results, without being aware of the limitations and liabilities involved. The fact that computers are usually used for simulation increases the hazard; to the uninitiated computers suggest an infallibility that does not in fact exist.

Another problem is that many people expect "answers" from a simulation. True, a **properly planned** simulation experiment, using a model that has been completely **validated for the purpose for which it is being used,** can yield answers. But these conditions are not often met. Nor need they be for a simulation to be **useful.** The value of simulation lies much more often in the insight that the effort imparts than in the answers that the results suggest.



"It's simulation. But if the building moves, then it's something else—"

****

Note to Do-It-Yourselfers . . .

If you're looking for a good complete article on how to build an inexpensive logic laboratory get a copy of the December 1970 issue of RADIO ELECTRONICS magazine, turn to page 33 and begin.

RADIO ELECTRONICS is published by Gernsback Publications, Inc. at 200 Park Ave., South, New York, 10003. Subscription Service – Boulder, Colorado 80302.

NEW REFORMATION:
Notes of a Neolithic Conservative

Paul Goodman
Vintage Books V–121
$1.95

Contents:

Preface
Part One: Science and Professions
Part Two: Education of the Young
Part Three: Legitimacy
Notes of a Neolithic Conservative

Many young people who find themselves fascinated with technology also feel nagging qualms about being involved with the products of corporate technology. Goodman provides some very lucid discussions of technology and eithics. The section on education also has important ideas that need to be considered by students and teachers. Goodman's personal, almost conversational style of writing makes time spent with this book enjoyable as well as thought provoking.

In automating, there is an analogous dilemma of how to cope with masses of people and get economies of scale without losing the individual at great consequent human and economic cost. A question of immense importance for the immediate future is, Which functions should be automated or organized to use business machines, and which should not? This question also is not getting asked, and the present disposition is that the sky is the limit for extraction, refining, manufacturing, processing, packaging, transportation, clerical work, ticketing, transactions, information retrieval, recruitment, middle management, evaluation, diagnosis, instruction, and even research and invention. Whether the machines can do all these kinds of jobs and more is partly an empirical question, but it also partly depends on what is meant by doing a job. Very often, for example in college admissions, machines are acquired for putative economies (which do not eventuate), but the true reason is that an overgrown and overcentralized organization cannot be administered without them. The technology conceals the essential trouble, perhaps that there is no community of the faculty and that students are treated like things. The function is badly performed, and finally the system breaks down anyway. I doubt that enterprises in which interpersonal relations are very important are suited to much programming.

But worse, what can happen is that the real function of an enterprise is subtly altered to make it suitable for the mechanical system. (For example, "information retrieval" is taken as an adequate replacement for critical scholarship.) Incommensurable factors, individual differences, local context, the weighing of evidence, are quietly overlooked, though they may be of the essence. The system, with its subtly transformed purposes, seems to run very smoothly, it is productive, and it is more and more out of line with the nature of things and the real problems. Meantime the system is geared in with other enterprises of society, and its products are taken at face value. Thus, major public policy may depend on welfare or unemployment statistics which, as they are tabulated, are not about anything real. In such a case, the particular system may not break down; the whole society may explode.

I need hardly point out that American society is peculiarly liable to the corruption of inauthenticity. Busily producing phony products, it lives by public relations, abstract ideals, front politics, show-business communications, mandarin credentials. It is preeminently overtechnologized. And computer technologists especially suffer the euphoria of being in a new and rapidly expanding field. It is so astonishing that a robot can do the job at all, or seem to do it, that it is easy to blink at the fact that he is doing it badly or isn't really doing quite the job.

* * * * * * * *

My books are full of one-paragraph or two-page "histories"—of the concept of alienation, the system of welfare, suburbanization, compulsory schooling, the anthropology of neurosis, university administration, citizenly powerlessness, missed revolutions, etc., etc. In every case my purpose is to show that a coerced or inauthentic settling of a conflict has left an unfinished situation to the next generation, and the difficulty becomes more complex in the new conditions. Then it is useful to remember the simpler state before things went wrong; it is hopelessly archaic as a present response, but it has vitality and may suggest a new program involving a renewed conflict. This is the therapeutic use of history. As Ben Nelson has said, the point of history is to keep old (defeated) causes alive. Of course, this reasoning presupposes that there is a nature of things, including human nature, whose right development can be violated. There is.

An inauthentic solution complicates, produces a monster. An authentic solution neither simplifies nor complicates, but produces a new configuration, a species, adapted to the on-going situation. There is a human nature, and it is characteristic of that nature to go on making itself ever different. This is the humanistic use of history, to remind of man's various ways of being great. So we have become mathematical, tragical, political, loyal, romantic, civil-libertarian, universalist, experimental-scientific, collectivist, etc., etc.—these too accumulate and become a mighty heavy burden. There is no laying any of it down.

The complement to prudent technology is the ecological approach in science. To simplify the technical system and modestly pinpoint our artificial intervention in the environment is to make it possible for the environment to survive in its complexity, evolved for a billion years, whereas the overwhelming instant intervention of tightly interlocked and bulldozing technology has already disrupted many of the delicate sequences and balances. The calculable consequences are already frightening, but of course we don't know enough, and won't in the foreseeable future, to predict the remote effects of much of what we have done.

Cyberneticists come to the same cautious thinking. The use of computers has enabled us to carry out crashingly inept programs on the basis of willful analyses; but we have also become increasingly alert to the fact that things respond, systematically, continually, cumulatively; they cannot simply be manipulated or pushed around. Whether bacteria, weeds, bugs, the technologically unemployed, or unpleasant thoughts, we cannot simply eliminate and forget them; repressed, they return in new forms. A complicated system works most efficiently if its parts readjust themselves decentrally, with a minimum of central intervention or control, except in cases of breakdown. Usually there is an advantage in a central clearing house of information about the gross total situation, but technical decision and execution require more minute local information. The fantastically rehearsed Moon landing hung on a last-second correction on the spot. To make decisions in headquarters means to rely on information from the field that is cumulatively abstract and may be irrelevant, and to execute by chain-of-command is to use standards that cumulatively do not fit the abilities of real individuals in concrete situations.

* * * * * * * *

12. The right style in planning is to eliminate the intermediary, that which is neither use, nor making for use. We ought to cut down commutation, transport, administration, overhead, communications, hanging around waiting. On the other hand, there are very similar functions that we ought to encourage, like travel and trade, brokering, amenity, conversation, and loitering, the things that make up the busy and idle city, celebrated by Jane Jacobs. The difference seems to be that in logistics, systems, and communications, the soul is on ice till the intermediary activity is over with; in traffic, brokering, and conversation, people are thrown with others and something might turn up. It is the difference between urbanism that imperially imposes its pattern on city and country both and the city planning for city squares and shops and contrasting rural life.

It was the genius of American pragmatism, our great contribution to world philosophy, to show that the means define and color the ends, to find value in operations and materials, to dignify workmanship and the workaday, to make consummation less isolated, more in-process-forward, to be growth as well as good. But in recent decades there has occurred an astonishing reversal: the tendency of American philosophy, e.g. analytic logic or cybernetics, has been to drain value from both making and use, from either the working and materials or moral and psychological goods, and to define precisely by the intermediary, logistics, system, and communications, what Max Weber called rationalization. Then the medium is all the message there is. The pragmatists added to value, especially in everyday affairs. Systems analysis has drained value, except for a few moments of collective achievement. Its planning refines and streamlines the intermediary as if for its own sake; it adds constraints without enriching life. If computation makes no difference to the data or the outcome—"Garbage in, garbage out"—then, to a pragmatist, the computation adds to the garbage. In fact, the computation abstracts from the data what it can handle, and constrains the result to what it can answer. Certainly, cybernetics could be enriching, as psychiatry or as ecology, but it has not yet been so—an exception has been the work of Bateson.

It is interesting to notice the change in the style of scientific explanation. At the turn of the century they spoke of development, struggle, coping, the logic of inquiry. Now they emphasize code, homeostasis, feedback, the logic of structure.